Clube Delphi + PHP

DevMedia group

Ano 9 · Edição 104 · R$11,90

Chaves em ClientDataSet

Você sabe utilizar corretamente chaves primárias e estrangeiras em sua aplicação?

Ask the Expert
Destacando uma série no TChart

QuickUpdate
PDF/A, você sabe o que é isso?

DataSnap e SOAP
Crie aplicações integradas com duas das principais tecnologias existentes no mercado

Boas Práticas
Realize testes unitários automatizados usando o DUnit

Boas Práticas
Aprenda a utilizar padrões de projeto em suas aplicações

EasyDelphi
Utilize todo o poder do Google Maps em seus sistemas Win32

EasyDelphi
Desenvolva facilmente gráficos utilizando informações do banco de dados

PHP
Compacte arquivos on-line utilizando o PHP

ISSN 1517990-7

Na Web :

Confira no portal ClubeDelphi PLUS 3 vídeo-aulas:

- Interagindo com WebCam no Delphi
- Trabalhando com diretivas de compilação
- Conheça o plug-in CanTools

Engenharia de Software
CONFERENCE

22 e 23 de maio – São Paulo

Raras são as oportunidades de ter acesso à informações
que podem transformar sua carreira. Essa é uma delas.

Reserve sua agenda. Inscrições limitadas.
www.devmedia.com.br/es_conference

Maiores informações através do e-mail evento@devmedia.com.br
ou pelo telefone (21) 3382-5025

DevMedia group

Sumário

Olá, eu sou o **DevMan**! Desta página em diante, eu estarei lhe ajudando a compreender com ainda mais facilidade o conteúdo desta edição. Será um prazer contar com sua companhia! Confira abaixo o que teremos nesta revista:

Delphi

Quick Update

08 – PDF/A, você sabe o é isso?
[*Paulo Quicoli*]

Tutorial

12 – Sistemas DataSnap com SOAP – Parte II
Integre seus sistemas utilizando duas das principais tecnologias
[*Maikel Scheid e Edinei Daniel Steffen*]

Delphi Expert

22 – Chaves no ClientDataSet
Geração de chaves com master-detail e ClientDataSet
[*Manoel Edésio*]

Boas Práticas

Boas Práticas, Projeto/Análise

30 – Padrões de Projeto
Entenda o que são Design Patterns e como aplicá-los a seu projeto
[*Paulo Quicoli*]

Boas Práticas, Projeto/Análise

35 – Testes Automatizados
Exemplos práticos usando o DUnit
[*Paulo Quicoli*]

Expert

43 – Editor de fórmulas lógico-matemáticas
Monte um sistema para criação de fórmulas matemáticas
[*Elvis Fusco*]

Easy Delphi

Easy Delphi

49 – Google Maps com Win32
Integre seus sistemas com Google Maps
[*Adriano Santos*]

Easy Delphi

55 – Usando dados do BD para gerar Gráficos
Veja como gerar gráficos usando informações do seu banco de dados
[*Ricardo Scache Belardinuci*]

PHP

PHP

63 – Compactando com PHP
Saiba como compactar arquivos usando o PHP
[*Adriano Santos*]

Você percebeu os ícones ao lado de cada matéria? Eles indicam o que você vai encontrar no artigo – dessa forma, você também pode ter uma idéia geral do que vai encontrar nesta edição como um todo! Os editores trabalham sempre no sentido de fechar a revista seguindo esta definição, para oferecer a você o melhor conteúdo didático!

Confira abaixo a lista com a definição dos tipos de artigo encontrados nesta edição:

[Mão na Massa] Artigos que focam na resolução de problemas - e não na tecnologia. A tecnologia empregada é apenas consequência. O artigo parte do pressuposto que o leitor já conhece a tecnologia, mas tem dificuldade de implementá-la em aplicação cotidiana. Por exemplo, um artigo que ao invés de "debulhar" todos os métodos e propriedades de um "recordset", mostre como usar o recordset de forma inteligente para criar um carrinho de compras.

[PHP] Artigo com foco no PHP puro.

[Expert] Artigo com foco no leitor avançado.

[Boas Práticas] Um dos objetivos da revista é levar para o leitor não somente as melhores técnicas mas também as melhores práticas. Esse tipo de artigo foca em técnicas que poderão aumentar a qualidade do desenvolvimento de software.

[Web] Artigos sobre/ ou que envolvam técnicas de desenvolvimento para Web.

[Novidades] Artigos sobre tecnologias de ponta, que ainda não fazem parte o dia-a-dia do desenvolvedor mas que prometem ser a próxima febre.

[Boa Idéia] A esse tipo de artigo é quase um artigo "Mão na Massa" - a diferença é que o exemplo do artigo é tão criativo que é considerado uma "Boa Idéia", dentro do contexto do desenvolvimento de software.

[Easy Delphi] Com foco no desenvolvedor iniciante.

Clube Delphi +PHP

Ano 9 - 104ª Edição 2009 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Editor Geral
Adriano Santos
adriano.santos@devmedia.com.br

Editor Técnico
Paulo Quicoli
pauloquicoli@devmedia.com.br

Comissão Editorial
Fabrício Desbessel, Paulo Quicoli, Adriano Santos e Guinther Pauli.

Diagramação
Romulo Araujo
romulo@devmedia.com.br

Revisão e Supervisão
Thiago Vincenzo - thiago.v.ciancio@devmedia.com.br

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Carmelita Mulin – Atendimento ao Leitor
www.devmedia.com.br/central/default.asp
(21) 3382-5025

Kaline Dolabella – Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 3382-5025

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

Kaline Dolabella
publicidade@devmedia.com.br

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Adriano Santos - Editor da Revista
adriano.santos@devmedia.com.br

EDITORIAL

Você realmente sabe como utilizar chaves primárias e estrangeiras em uma aplicação? Por incrível que pareça, já vi muito desenvolvedor de longa data se atrapalhando com isso. Não é um assunto muito discutido é verdade, mas é um dos pontos principais de uma aplicação, seja ela de pequeno, médio ou grande porte. O Manoel Edésio se propôs a escrever um mega artigo falando do assunto, e se deu bem. Em seu artigo ele detalha a utilização de chaves e mostra como trabalhar com elas de maneira simples, porém eficaz.

Na sessão Boas Práticas você encontrará dois grades artigos do nosso Editor Técnico Paulo Quicoli. No primeiro, ele continua a série de artigos sobre Design Patterns, que é um dos assuntos mais discutidos em TI. A orientação a objetos é, sem dúvida, um assunto muito em pauta e fascina muito desenvolvedor. Dessa vez o Paulo fala dos padrões estruturais Adapter, Decorator e Façade. Vale a pena conferir. E já que estamos falando em padrões, porque não dar uma passadinha em testes unitários automatizados? Não dá para distribuir aplicações com bugs para todo o lado. Com o framework DUnit, é possível efetuar os mais variados e complexos testes em nossas aplicações. Veja com o Paulo como fazer isso.

Você já teve que desenvolver diversas fórmulas matemáticas e utilizá-las várias vezes ao longo da aplicação? Claro que já, com certeza. ISS, INSS, FGTS, ICMS, enfim, são todas fórmulas que precisam ser utilizadas no dia-a-dia. E se eu te dissesse que é possível criar suas próprias fórmulas e armazená-las no banco de dados para utilização a qualquer hora? O Elvis nos mostra um componente que nos estende essa possibilidade. O componente TExpressionParser tem essa finalidade. Veja como é fácil manipular fórmulas com ele.

A onda do momento é navegar no Google Maps, pois é. Com ele não estamos mais sozinhos, é incrível como tem ajudado. Eu estudei pessoalmente a API do Google Maps e descobri inúmeros recursos. O legal é que podemos utilizá-lo gratuitamente e em aplicações Win32. Veja como é fácil incluir apontadores de rotas e mapas em seus sistemas. Em outro artigo na sessão Easy-Delphi, o Ricardo mostra como gerar gráficos usando informações do banco de dados.

Por fim, você vai conferir no meu artigo de PHP como compactar arquivos on-line utilizando a biblioteca ZLib. Ela está presente no phpMyAdmin e é bem simples de ser utilizada. Confira.

Obrigado, uma boa leitura e bons códigos. Até a próxima.

Adriano Santos

adriano.santos@devmedia.com.br

Editor Chefe da revista Clube Delphi
Conheça nosso BLOG em *clubedelphiphp.blogspot.com*

ClubeDelphi PLUS

DevMedia group

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS.
Para mais informações sobre o pacote PLUS, acesse:
http://www.devmedia.com.br/clubedelphi/portal.asp

ClubeDelphi PLUS

Informativo ClubeDelphi

Portal ClubeDelphi

www.clubedelphi.net/portal

+ 600 vídeo aulas e 7 cursos online

Brinde na web desta edição - 3 vídeos! Confira:

3 Vídeos

• **Interagindo com WebCam no Delphi**
Aprenda com o Adriano Santos a utilizar o componente TCamera-Delphi para interagir com sua WebCam, podendo gravar vídeos ou até mesmo tirar fotos.
http://www.devmedia.com.br/articles/viewcomp.asp?comp=7914

• **Trabalhando com diretivas de compilação**
Saiba o que são e como utilizar Diretivas de Compilação em seus sistemas com o autor Ricardo Boaro.
http://www.devmedia.com.br/articles/viewcomp.asp?comp=7248

• **Conheça o plug-in CanTools**
Adriando Santos mostra como turbinar a IDE do seu Delphi utilizando o plug-in CanTools.
http://www.devmedia.com.br/articles/viewcomp.asp?comp=5887

Para acessar os vídeos utilize os seguintes dados
Login: **DVM.PL** *Senha:* **xkm23v**

Gostou das vídeo aulas? O portal *www.devmedia.com.br* possui mais de **2 mil vídeo aulas** e **dezenas de cursos online** sobre desenvolvimento de software! Agora você pode comprar as vídeo aulas que preferir e fazer sua própria combinação de vídeos! Saiba mais em *www.devmedia.com.br/creditos*

Edições Anteriores da ClubeDelphi

Você pode comprar todas as edições anteriores da ClubeDelphi através do site da DevMedia!

Para isso basta acessar *https://seguro.devmedia.com.br/edicoes_anteriores.asp*

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Para votar, você vai precisar do código e senha de banca desta edição, que é:
Login: **DVM.PL** Senha: **xkm23v**

Perguntas e Respostas

Destacando a série selecionada em um TChart

Perguntas respondidas por Paulo Quicoli
pauloquicoli@devmedia.com.br

Nossa leitora Bianca Rodrigues enviou para nós uma dúvida sobre o componente *TChart*, que para quem ainda não o conhece é um componente que permite a criação de gráficos e é distribuído junto com o *Delphi*. A situação dela é a seguinte: em seu sistema existe um gráfico de barras que permite ao usuário selecionar alguma das barras e realizar uma tarefa específica, porém essa barra precisaria ficar em destaque, informando ao usuário que está selecionada.

Muito bem, vamos te ajudar nessa. Crie um novo projeto no *Delphi*, adicione um componente *TChart* no formulário, adicione uma série vazia ao gráfico e vamos definir seu visual clicando com o botão direito do mouse sobre ele. No menu que surgir, escolha *Edit Chart*. Selecione a aba *Series* e nela selecione *General*. Na aba *General* desmarque o *CheckBox Show in Legend*. Clique em *Marks*, no *ComboBox Style* escolha o item *Value*. Agora vá à aba *Gradient* e marque o *ComboBox Visible*. Vamos também mudar o título desse gráfico. Clique na aba *Chart* e depois em *Titles* e defina o título como *Visitantes do meu site*. Para este exemplo vamos popular o gráfico com alguns dados fictícios, a quantidades de visitas que um *site* recebeu nos últimos cinco meses. Cada informação criará uma barra no gráfico e cada barra terá uma cor diferente.

Para criar as cores no gráfico vamos criar um campo privado no formulário chamado *Cores* do tipo *Array of TColor*, como no código a seguir.

```
cores : array of TColor ;
```

Esse *array* vamos utilizar em uma procedure que o Wpreenche com as cores que precisamos, como pode ser vista na **Listagem 1**.

Listagem 1. Criando as cores

```
01 procedure TForm1.CarregarCores;
02 begin
03  SetLength(cores, 6);
04  cores[0] := clBlack;
05  cores[1] := clGreen;
06  cores[2] := clRed;
07  cores[3] := clMaroon;
08  cores[4] := clNavy;
09 end;
```

Depois vamos criar outra procedure que irá inserir os dados fictícios no gráfico, como mostra a **Listagem 2**.

Listagem 2. Criando as informações

```
01 procedure TForm1.CarregarDadosNoGrafico;
02 begin
03  Chart1.Series[0].Clear;
04  Chart1.Series[0].Add(33,'Janeiro',cores[0]) ;
05  Chart1.Series[0].Add(72,'Fevereiro',cores[1]) ;
06  Chart1.Series[0].Add(95,'Março',cores[2]) ;
07  Chart1.Series[0].Add(41,'Abril',cores[3]) ;
08  Chart1.Series[0].Add(87,'Maio',cores[4]) ;
09 end;
```

Com essas duas *procedures* definidas, no evento *onCreate* do formulário chamamos primeiramente *CarregaCores* e em seguida *CarregarDadosNoGrafico*.

Para destacar, ao clicar do mouse, alguma coluna da série é preciso implementar o evento *onClickSeries* do *TChart*. Dê um clique duplo nesse evento e adicione o código da **Listagem 3.**

Esse evento ao ser disparado nos informa qual valor da série foi clicado através da variável *ValueIndex*. A série clicada está representada na variável *Series*. De posse dessas duas informações apenas passamos a nova cor para o valor da série clicado através do método *ValueColor*. Veja o resultado na **Figura 1**. Experimente compilar o exemplo e testar clicando nas colunas do gráfico. No exemplo da Figura 1, cliquei na 4º coluna que ficou. ●

Listagem 3. Destacando a coluna clicada

```
01 procedure TForm1.Chart1ClickSeries(Sender:
    TCustomChart;
02   Series:TChartSeries;
03   ValueIndex: Integer; Button: TMouseButton;
04   Shift: TShiftState; X, Y: Integer);
05 begin
06   CarregarDadosNoGrafico;
07   Series.ValueColor
      [ValueIndex] := clYellow;
08 end;
```

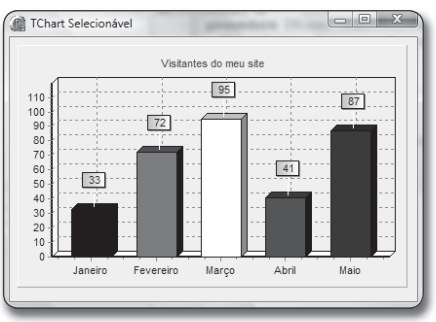


Figura 1. TChart indicando barra selecionada

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link: ***www.devmedia.com.br/clubedelphi/feedback***

Hospede com quem fala sua língua!

Local host

Hospedagem Profissional
Revenda
Parcerias

Ativação imediata!

Faça sua assinatura, tenha 30 dias grátis

e seu primeiro boleto para 60 dias após assinatura em qualquer plano

UNIX I

a partir R$ 28,00

- 1 GB Espaço em disco
- 60 GB Transferência
- 30 Contas de e-mail
- 10 Domínios
- Painel de Controle em Português
- Atendimento exclusivo

Contratar

UNIX II

a partir R$ 58,00

- 3 GB Espaço em disco
- 120 GB Transferência
- 60 Contas de e-mail
- 15 Domínios
- Painel de Controle em Português
- Atendimento exclusivo

Contratar

UNIX III

a partir R$ 98,00

- 6 GB Espaço em disco
- 300 GB Transferência
- 90 Contas de e-mail
- 20 Domínios
- Painel de Controle em Português
- Atendimento exclusivo

Contratar

R$ 5,90 — **Mini html**
Tráfego 1 GB
Espaço 100 MB
3 C.Ppostal, Espaço 3G
Domínio 3

R$ 15,90 — **html**
Tráfego 25 GB
Espaço 500 MB
20 C.Ppostal, Espaço 20G
Domínio Ilimitado

R$ 15,90 — **ASP**
Tráfego 25 GB
Espaço 500 MB
20 C.Ppostal, Espaço 20G
Domínio Ilimitado
MySql 50m

R$ 20,90 — **PHP**
Tráfego 25 GB
Espaço 500 MB
20 C.Ppostal, Espaço 20G
Domínio Ilimitado
MySql 50m

Formato exclusivo de trabalho para empresas de desenvolvimento de sites, websites, portais, serviços e negócios que necessitam de grandes tráfegos, monitoramento e backup.

www.lhost.com.br

assinar@lhost.com.br

Nesta seção você encontra artigos intermediários sobre Delphi Win 32 e Delphi for .NET

Quick Update

PDF/A, você sabe o é isso?

Paulo Quicoli
pauloquicoli@devmedia.com.br
Editor técnico da revista ClubeDelphi, formado em processamento de dados pela FATEC-TQ. Atua como analista e desenvolvedor da ControlM Informática (www.controlm.com.br). Utiliza Delphi desde de sua primeira versão, entusiasta do desenvolvimento orientado a objetos tem publicado vários artigos sobre o assunto. Blog: www.pauloquicoli.spaces.live.com .

Para quem ainda não percebeu, mesmo estando numa época onde *tudo* é digital continuamos cada vez mais acumulando papel em nossas gavetas, arquivos, escritórios. Tudo isso ocupa muito espaço ao longo do tempo. Empresas precisam manter um histórico de seus documentos contábeis por alguns anos e quando algum *documento antigo* é solicitado, se pegam perdidos procurando por entre caixas e caixas. A tecnologia hoje oferece um atenuante para esse problema, os sistemas especialistas em *GED* (Gerenciamento Eletrônico de Documentos).

Esses sistemas tem a missão de digitalizar esses documentos importantes e indexá-los de tal forma que se torne fácil sua visualização, localização e reprodução ao longo dos anos. No Brasil isso já uma realidade. Em diversos cartórios para se realizar um reconhecimento de firma não se buscam mais as *fichas* e sim a imagem digitalizada dela em um sistema *GED*. Até mesmo para uso interno empresas utilizam esse recurso para localizar *aquele* pedido feito por um cliente, ou *aquela* Nota Fiscal.

Nos últimos anos esse é um mercado crescente que até então utilizava o que era o mais adequado. Os documentos digitalizados eram armazenados no formato de imagem *.TIFF*. Esse formato permite uma boa reprodução da imagem quando necessário e por ter uma compactação superior, resolvia bem um problema novo criado pelo *GED:* o espaço em disco para armazenar todos os documentos digitalizados.

Porém a *internet* e a mobilidade chegaram e com elas novos requisitos ao *GED* foram solicitados. Esses documentos digitalizados agora precisam estar disponíveis na *internet*. Claro que não para todos, mas para quem interessa e tem permissão de visualizar. Esse foi o desafio encontrado, como vamos compartilhar a visualização um arquivo *.TIFF* na *internet*, já que os navegadores não oferecem suporte a esse formato.

A solução encontrada foi a geração de arquivos *.PDF* a partir da imagem original em *.TIFF*. Com isso, o formato *.PDF* foi tornado-se o padrão para esse tipo de função. Por possuir, em muitos casos, compactação superior ao *.TIFF* e oferecer recursos mais interessantes como armazenamento de metadados (informações como autor, título, palavras-chave dentro da estrutura do arquivo) que podem ser indexados sem intervenção humana sua utilização na *internet* foi crescente e garantiu a popularidade do formato que vem se atualizando a cada ano, incluindo novos recursos. Com isso surgiu um novo problema, a compatibilidade entre os formatos *.PDF*.

Por exemplo, caso algum arquivo seja criado utilizando algum recurso do formato versão 2.0 como ele será visualizado por alguém que ainda possui o visualizador na versão 1.5? Desse grande impasse, surge então o *PDF/A*.

A Normatização

O *ISO* publicou uma norma, a *ISO 19.005-1*, que determina que o formato *.PDF* deve ser utilizado universalmente para armazenamento de arquivos digitais de longa duração. Essa medida foi criada para auxiliar órgãos governamentais e empresas que precisam garantir a visualização e reprodução de seu acervo digital para os próximos 30 ou 40 anos. Contudo a versão do formato *.PDF* normatizada não é qualquer uma. Foi criada uma especificação única e comum para esse tipo utilização. Esse formato é denominado *PDF/Archieve* ou simplesmente *PDF/A*.

O Diferencial

A norma *ISO 19.005-1* não rege como esses arquivos devem ser armazenados, como devem ser localizados. Ela determina os requisitos que o formato deve atender para garantir sua visualização e reprodução ao longo dos anos. O principal requisito é que o formato *.PDF/A* deve ser 100% auto-suficiente. Ou seja, tudo o que ele precisa para ser visualizado ou impresso está contido nele. Imagine que originalmente um arquivo *.PDF* teve seu texto criado com a fonte *Calibri*. Então esse arquivo foi enviado para uma outra pessoa que em seu computador não possui essa fonte. Ao visualizar o arquivo, o leitor utilizado irá substituir a fonte *Calibri* provavelmente pela *Arial*, descaracterizando assim o arquivo. Tornar o formato auto-suficiente garante que isso não aconteça, o arquivo será visto como na sua criação. Nisso incluem-se além das fontes utilizadas, informações sobre cores, imagens e tudo mais que possa ser utilizado para sua criação.

Apesar de ter sua estrutura na versão 1.4 do formato *.PDF*, alguns recursos foram retirados. Por exemplo efeitos de transparência, sons e vídeos não são permitidos.

Gerando PDF/A no Delphi

Talvez seu sistema *GED* gere arquivos *.PDF* comuns, ou ainda apenas digitalize as imagens no formato *.TIFF* ou, na pior das hipóteses, armazena no formato *.JPEG*. Se seu cliente ainda não solicitou que seu sistema esteja de acordo com a norma *ISO 19.005-1*, antecipe-se. Vamos ver como fazer isso agora.

Vou apresentar aqui uma solução que contempla os sistemas que digitalizam os documentos no formato *.TIFF* e posteriormente serão convertidos para o *.PDF/A*. A solução envolve utilização de componentes de terceiros, pagos.

Solução

Vamos utilizar a suíte *GDPicturePro*, que pode ser baixada em *www.gdpicture.com*. Essa suíte de componentes é muito completa. A geração de *.PDF/A* é apenas um dos seus recursos. Para poder ser utilizada no *Delphi* é necessário baixar a versão *ActiveX*, caso você esteja utilizando *.NET* existe versão específica em código gerenciado.

Nota: Não entrarei em detalhes de como instalar o componente, já que isso é muito bem explicado no arquivo *ReadMe* que é distribuído junto com o componente.

Listagem 1. Convertendo um arquivo .TIFF para o formato .PDF/A

```
01 procedure ExportarParaPdfA(ImagemTiff: string;
02                            ArquivoDestinoPdfA: string);
03 var
04   Imaging1  : TImaging;
05   nImageID: Integer;
06   GdViewer1 : TGdViewer;
07 begin
08   Imaging1 :=TImaging.Create(nil);
09   GdViewer1:=TGdViewer.Create(nil);
10   GdViewer1.DisplayFromFile(ImagemTiff);
11   if GdViewer1.GetStat = 0 then
12      Imaging1.SetNativeImage(GdViewer1.GetNativeImage);
13
14   Imaging1.SaveAsPDFA(ArquivoDestinoPdfA,
15                       'Teste',
16                       'DevMedia Group',
17                       'Paulo Quicoli',
18                       'GdPicture SDK 5.7.4');
19   Imaging1.CloseImage(nImageID);
20   FreeAndNil(Imaging1);
21   FreeAndNil(GdViewer1);
22 end;
```

A seguir temos algumas das funcionalidades disponíveis:

- Exibição de arquivos *.PDF* sem a necessidade de qualquer software visualizador externo;
- Conversão de arquivos *.TIFF multipage* para *.PDF*;
- Conversão entre formatos de imagens, por exemplo *.PSD* para *.TIFF*, *.PDF* para *.TIFF*, *.JPG* para *.TIFF*, e outras mais;
- Suporte às transparências existentes nas imagens;
- Aplicação de filtros como *Noise, Blur, Sharpen*, etc;
- Criação de marca d'água;
- Recursos de *OCR;*
- Digitalização de imagens através de *driver TWAIN*;
- Reconhecimento de códigos de barras em imagens;
- Impressão de imagens com recursos avançados.

Baseado nos exemplos disponíveis da suíte podemos converter um arquivo *.TIFF* criando a seguinte *procedure*, conforme **Listagem 1**.

Nas linhas 8 e 9 instanciamos os objetos da suíte responsáveis respectivamente por manipular as imagens e por carregar e/ou exibi-las. Depois disso carregamos então a imagem passada como parâmetro, linha 10, e se não ocorreu erro algum nisso informamos ao objeto *Imaging1* que ele deve manipular a imagem contida em *GdViewer1*, linha 12.

Então na linha 14 utilizamos o método *SaveAsPdfA* que recebe como parâmetros: o arquivo de destino, o título do arquivo, a empresa responsável pelo arquivo, o autor e a ferramenta utilizada para geração do arquivo. Esses últimos quatro parâmetros ficarão registrados nos metadados do arquivo de destino.

Faça o teste você mesmo. Baixe a suíte *GDPicture* e utilize não só o recurso apresentado. Mais detalhes sobre o formato *.PDF/A* podem ser encontrados em *www.pdfa.org*. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

A partir de agora você pode contar com a ajuda da

Chegou a Consultoria On-line DevMedia

Consultoria Técnica + Professor Virtual + Certificação

Mais Informações:
www.devmedia.com.br/consultoria_online 21 3382-5025

DevMedia em seus projetos e estudos.

A DevMedia possui um numeroso time de autores, editores e professores que juntos produzem o material que você está acostumado a encontrar em nosso site e revistas. E são exatamente esses mesmos profissionais que estarão a sua disposição para tirar suas dúvidas e ajudá-lo em seus projetos e estudos. Através de uma plataforma 100% web a Consultoria DevMedia garante sigilo absoluto, eficiência e rapidez em todas as respostas. Finalmente você terá ao seu alcance uma consultoria de qualidade por um preço muito acessível. Consulte nossos planos.

Mais um serviço DevMedia group

Nesta seção você encontra artigos intermediários sobre Delphi Win 32 e Delphi for .NET

Sistemas DataSnap com SOAP

Crie sistemas integrados usando SOAP – Parte II

Resumo do DevMan

O desenvolvimento de sistemas em n camadas tem se popupalirazado no cenário de TI. Fazer com que uma aplicação possa ser gerenciada a distância via Web não é uma tarefa simples, porém o que veremos é a construção de uma solução completa utilizando o protocolo SOAP e o conceito DataSnap. Veremos que é perfeitamente possível colocar tudo em prática em apenas alguns passos.

A estrutura que será apresentada dará um enfoque exclusivo no uso da tecnologia SOAP e DataSnap para a integração de sistemas entre empresas, tratando do funcionamento e regras de negócio com algumas preocupações centralizadas de segurança.

Nesse artigo veremos

- Implementação de uma estrutura SOAP e DataSnap para agilizar processos entre empresa e fornecedor;
- Apresentação de um cenário onde a mesma estrutura pode ser explorada;
- Desenvolvimento das camadas cliente.

Qual a finalidade

Demonstrar como fazer acesso aos dados utilizando a tecnologia SOAP.

Quais situações utilizam esse recurso?

Todo e qualquer sistema que necessita de visões on-line do cenário, podem ser construídos utilizando os conceitos que aprenderemos nesse artigo.

Maikel Marcelo Scheid
maikelscheid@gmail.com
É técnico em Informática com ênfase em Análise e Programação de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MSSQL.

Edinei Daniel Steffen
edineidaniel@gmail.com
É técnico e Bacharel em Sistemas de Informação, Analista e Desenvolvedor de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MSSQL.

Integração é o termo que iremos tratar nesta segunda e última etapa do artigo para soluções entre sistemas integrados. Como vimos na primeira etapa a construção da camada servidor, faremos agora a hospedagem da DLL de comunicação em um servidor *Web*. Usaremos o IIS (*Internet Information Services*), onde veremos um exemplo utilizando *Windows XP e IIS 6.0* e também *Windows Vista* com *IIS 7.0*. Desenvolveremos duas camadas de acesso a esses servidores. As duas aplicações das camadas cliente que iremos construir usarão esses servidores para acesso. Uma camada que fará o gerenciamento dos dados na empresa e a segunda camada, distribuída ao fornecedor, será responsável pela busca de pedidos realizados pelo cliente.

É importante lembramos que essa

tecnologia nos abre um grande leque de possibilidades. Integrar sistemas utilizando *DataSnap* e *SOAP* nos possibilita acesso rápido e seguro, além de nos possibilitar externar os dados de nosso *BD* para que outras empresas parceiras ou não, possam ter acesso e até *imputar* dados no sistema. Por exemplo: é comum a troca de arquivos entre empresa e banco para emissão de boletos bancários. São os famosos *CNAB's*. Da mesma forma existem empresas que trocam dados entre si utilizando-se de *EDI* (*Eletronic Data Interchange*). Em outras palavras é o uso de arquivos no formatado texto. Disponibilizando um sistema via *SOAP*, basta conhecer as funções disponíveis ao mundo externo e então fazer uso delas, seja com *Delphi*, *C*, *Java* ou qualquer outra linguagem.

Publicando a DLL no Windows XP + IIS 6.0

A hospedagem de sistemas com essa finalidade de integração entre *cliente/fornecedor* deverá ser em estruturas de servidores públicos à *Internet* para que o acesso seja possível tanto para empresa quanto para a lista de fornecedores que receberão o sistema. Tomaremos as devidas precauções com segurança de acesso, é claro. Nestes ambientes de servidores com sistemas operacionais *Windows XP, Windows 2000* e *2003 Server*, a seqüência de publicação de uma *DLL* é a mesma de acordo procedimento que veremos adiante.

Nota: O *IIS* (*Internet Information Services*) não é instalado por padrão no Windows. Caso não tenha optado por ele durante a instalação do Windows, será necessário entrar no item Adicionar ou Remover Programas no Painel de Controle e efetuar a instalalão.

Nota do DevMan

Os bancos usam o padrão Febraban (Federação dos Bancos) CNAB 400 ou 240 para receber (remessa) e enviar (retorno) informações para as empresas clientes usando arquivos. O objetivo destes arquivos é intercambiar informações digitalmente entre o sistema de informática do banco e o do cliente. Dentre as informações podemos citar: cobrança (boletos bancários), pagamentos, extrato (para conciliação), débito em conta, vendor e custódia de cheques. Cada um destes produtos tem seu fluxo de informação e portanto um layout.

O arquivo contém texto puro e colunas fixas definidas pela FEBRABAN, cada banco, por possuir suas próprias peculiaridades, tem suas variações que são previstas dentro do padrão.

Estes arquivos são enviados ou recebidos de diversas formas, mas principalmente pelo site do banco. Para exemplificar usaremos o intercambio de informações de boletos bancários:

- A empresa emissora dos boletos envia opcionalmente um arquivo remessa usando o site do banco. Esta etapa não é necessária, pois a própria empresa pode emitir os boletos;
- Os clientes recebem e pagam os boletos;
- A empresa verifica diariamente no site do banco a existência de arquivos remessa;
- A empresa processa no software financeiro os arquivos baixados no site do banco, ao processar o software faz a baixa dos boletos que foram liquidados, deixando pendente os outros.

Para acessar o *IIS*, acesse o *Painel de Controle>Ferramentas Administrativas>Gerenciador do Serviços de Informações da Internet (IIS).* A sua esquerda aparecerá uma árvore de opções. Encontre a opção *Sites da Web* e dentro dela *Default Web Site*. Usaremos a opção de um novo diretório virtual possibilitando que você hospede a DLL em qualquer diretório do seu computador. Com o botão direito do mouse sobre a opção *Default Web Site*, selecione a opção *Novo>Diretório virtual* onde surgirá um *Wizzard* de configuração do diretório virtual. Ao avançar a primeira tela digite um apelido, ou seja, o *alias pedidos* para o diretório e na tela seguinte selecione o diretório onde está hospedada a DLL *C:\Inetpub\scripts*. Na tela de permissões de acesso, deixe selecionadas as opções *Leitura*, *Executar scripts* e *Executar (aplicativos ISAPI ou CGI)*, sendo este último o responsável pela interpretação e execução da DLL (**Figura 1**). Realizado este processo das configurações confirme a tela e terá o diretório virtual criado. A partir do mesmo já poderá visualizar as funções disponíveis na *DLL* via *browser*. Para acessar o sistema via browser de agora em diante basta digitar o endereço a seguir:

```
http://localhost/pedidos/PedidosSoapServer.dll
```

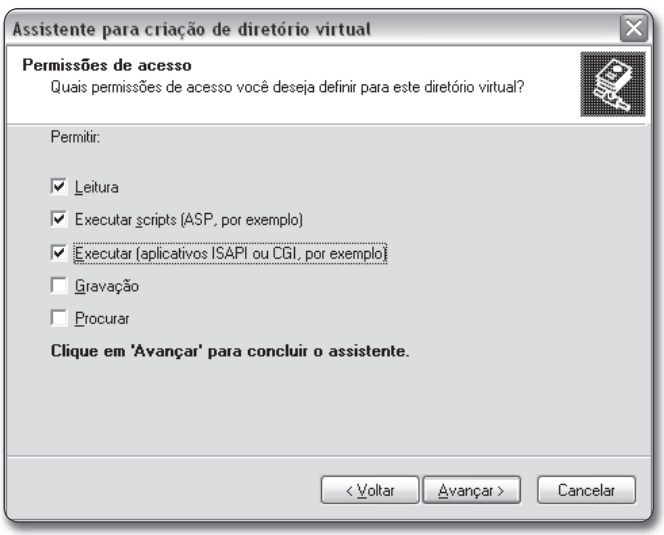


Figura 1. Permissões do IIS 6.0

Ao executar o endereço do diretório virtual criado através do *browser* da Internet, uma página contendo as informações do serviço é exibida (**Figura 2**). Nesta página são publicados todos os métodos disponíveis na *DLL*, caminho de referencias e endereço do *SOAP*.

Publicando a DLL no Windows Vista + IIS 7.0

Os novos sistemas operacionais buscam sempre maneiras mais eficiente para controlar a segurança de ambientes mais inteligentes e com maior melhor interação com usuários. Uma destas ferramentas é o *IIS 7.0* dos sistemas operacionais mais recentes, *Windows Vista* e *Windows Server 2008*. Utilizando a *DLL* do serviço *SOAP* criada, veremos a sua hospedagem na *Web*.

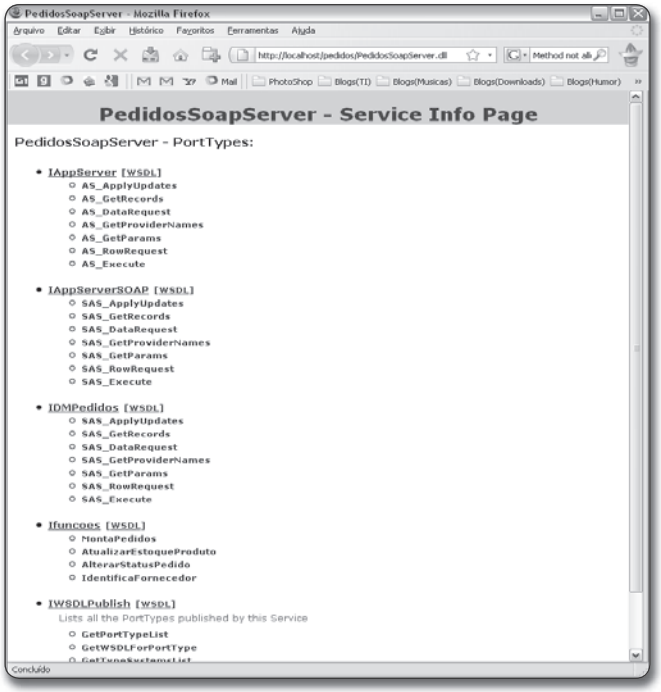


Figura 2. Página de informações do serviço

O caminho para acesso ao *IIS 7.0*, é o mesmo que vimos no tópico anterior. É importante ter previamente habilitado e instalado as funcionalidades de execução de *scripts CGI* e *ISAPI*. Iniciaremos com a criação de um diretório *scripts* em *C:\Inetpub*, local para onde iremos copiar a *DLL* criada. Após isso, utilizando o menu *Iniciar>Executar* digite *inetmgr* ou através do *Painel de Controle* abra o *console de gerenciamento do IIS*.

Semelhante ao processo do *IIS 6*, clique com o botão direito sobre *Default Web Site* e escolha a opção *Adicionar Aplicativo* (**Figura 3**), que trará um gerenciador. Digite o *alias pedidos*, mantendo o *Pool de Aplicativos* como *DefaultAppPool* e definindo o caminho físico onde estamos hospedando o arquivo de *DLL* (*C:\Inetpub\Scripts*) e confirme.

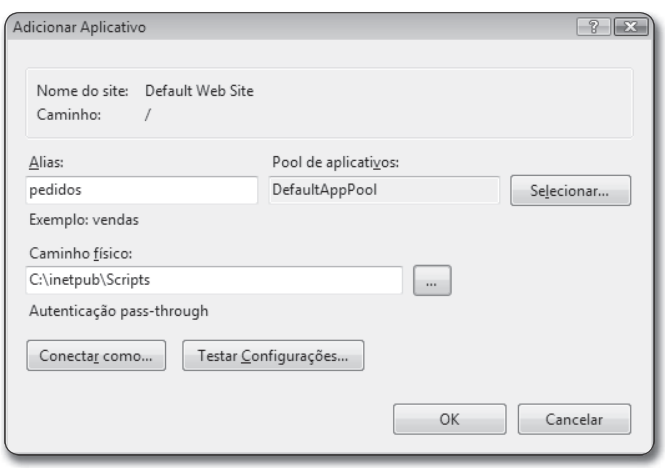


Figura 3. Adicionando novo aplicativo

Criado o aplicativo, precisaremos adicionar um novo *mapa de script*, ou seja, adicionar um novo manipulador *Web* funcionando como um mecanismo que ira responder por solicitações específicas, como ocorre no caso de *DLL's* e executáveis. Selecionando agora o aplicativo *pedidos* criado, observe ao centro do console de configurações do *IIS* um ícone chamado *Mapeamentos do Manipulador.* Acesse-o a fim de criar a modificação. Com o botão direito do mouse em qualquer parte, selecione a opção *Adicionar Mapeamento de Script*, preenchendo as informações de acordo com a **Figura 4**.

Conforme o próprio *help* do *IIS* nos mostra, em *caminho da solicitação* digite uma extensão de nome de arquivo com uma extensão para a qual o manipulador deve processar solicitações que será o nosso caso. O manipulador deverá executar todas as solicitações feitas para um arquivo específico, portanto digite o nome do arquivo e a extensão, por exemplo, *PedidosSoapServer.dll*.

Para o executável, selecione a *DLL* no diretório que foi hospedada e atribua também um nome (*pedidos*). Confirme as configurações da tela e confirme também a mensagem indicando a adição de um novo *script.* Ainda com o aplicativo *pedidos*, selecionado na aba esquerda do console, selecione no rodapé do mesmo a opção *Exibição de Conteúdo*, listando assim a *DLL* relacionada.

Clique com o botão direito sobre a mesma escolhendo a opção *Procurar*, método que fará com que o *browser* padrão de internet seja executado carregando a página de informações do arquivo, concluindo assim a etapa de configuração.

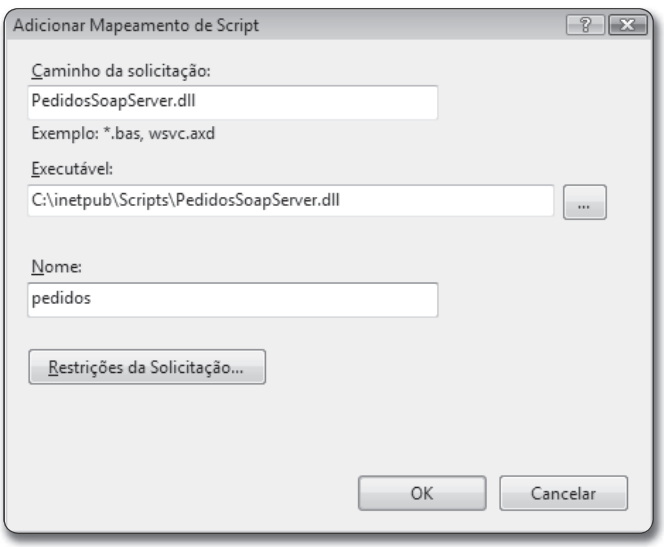


Figura 4. Adicionando mapeamento de script no IIS 7.0

Criando a aplicação cliente da empresa

Quando nos referimos à aplicação cliente da empresa, queremos dizer que será a aplicação responsável por gerenciar os pedidos no lado do servidor *SOAP*, na empresa. Será esta a aplicação que irá *cadastrar fornecedores, produtos, consultar o status de solicitações realizadas, itens, quantidades*, entre outros.

No *RAD Studio 2007*, através do menu *File>New>VCL Forms Application – Delphi for Win32* crie uma nova aplicação alterando inicialmente o nome do formulário principal para *frmMenu* e

sua propriedade *Caption* para *Terminal Cliente - Gerenciamento Sistema Pedidos*. Adicione logo em seguida também um componente *MainMenu(mmAcessos)* e com um duplo clique sobre o mesmo crie a estrutura representada pela **Figura 5**, que demonstra todas as funcionalidades a serem construídas na aplicação.

Após isso salve os arquivos em um diretório, atribuindo o nome *untMenu.pas* à *Unit* do formulário principal e *prjSOAP_TerminalCliente.dpr* ao projeto da aplicação.

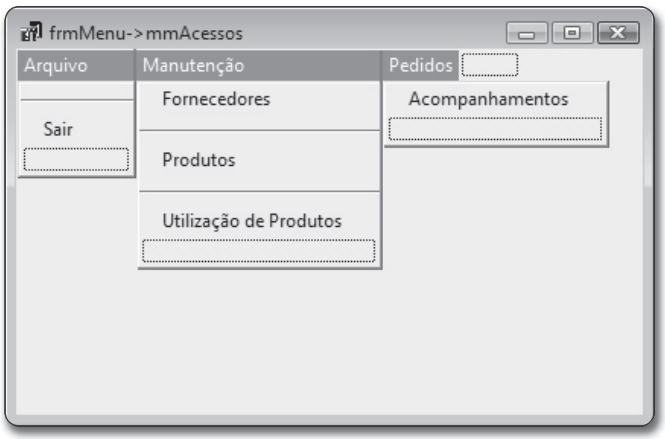


Figura 5. Estrutura de menu

Neste exemplo cliente, para comunicação com a camada servidora, usaremos dois processos distintos. Um deles para facilitar a chamada de métodos e outro para possibilitar uma fácil ligação entre componentes desta camada cliente com componentes de ligação a dados da camada *Server*. Primeiramente faremos então a importação da *WSDL* (*Web Service Definition Language*) que são as regras e codificações criadas e publicadas na *Web* através da *DLL*. O endereço da *WSDL* a ser importado é obtido após da configuração do *IIS*. Use o endereço de nossa aplicação conforme a seguir:

```
http://localhost/Pedidos/PedidosSoapServer.dll/wsdl/Ifuncoes
```

Com o endereço armazenado em memória, através do menu *File>New>Other*, selecione na guia *Delphi Projects* o item *WebServices* e em seguida selecione o ícone *WSDL Importer* e confirme. Na janela que surgir, cole o endereço armazenado em memória e avance visualizando assim um escopo da estrutura a ser importada. Confirme a tela e observe a criação de uma nova *unit* vinculada ao projeto (*ifuncoes.pas*), contendo os métodos de comunicação *SOAP* com o servidor do aplicativo, sendo eles o método *pai GetIfuncoes()* e os *sub-métodos* para montagem de pedidos, atualização de estoque, entre outros.

Depois de realizada a importação da *WSDL*, adicionaremos agora a aplicação um *Data Module (dmControle)* através do menu *File>New>Other* e na guia *Delphi Projects>Delphi Files* selecione a opção desejada, salvando-o em seguida como *untControle.pas*. Nesse processo faremos a utilização do endereço da *WSDL* através de componentes, e para iniciar, arraste para o formulário da paleta *WebServices* o componente *SoapConnection (ConexaoSOAP)* que fará a comunicação direta com o protocolo,

Nota do DevMan

Electronic Data Interchange - EDI significa troca estruturada de dados através de uma rede de dados qualquer.

EDI pode ser definido como o movimento eletrônico de documentos padrão de negócio entre, ou dentro, de empresas. O EDI usa um formato de dados estruturado automático que permite que os dados sejam transformados sem serem reintroduzidos.

Além disso, podemos considerar que o uso primário do EDI é transferir transações de negócio repetitivas tais como: encomendas, faturas, aprovações de crédito e notificações de envio. Isto significa que o EDI hoje, contrariamente ao que muitos acreditam, não implica comunicação em tempo real.

O termo EDI tradicional é usado para denotar o EDI com suporte para alguns dos padrões EDI, tais como EDIFACT ou ANSI X12, ou subconjuntos deles.

O EDI é sem dúvida potenciador para a comunicação de negócio efetiva e eficiente e na realidade ninguém se opõe à idéia de comunicação eletrônica entre organizações.

Podemos apontar os seguintes pontos fortes do EDI: É um padrão aberto e trans-setorial com fluxos de dados formalizados, garante a troca segura de dados, segura na perspectiva de que diferentes checksums garantem que os dados enviados são fidedignos.

sendo seu funcionamento semelhante a um componente de ligação nativa a banco de dados. Adicione a sua propriedade *URL* o caminho de hospedagem do serviço *SOAP* através do seguinte endereço:

```
http://localhost/Pedidos/PedidosSoapServer.dll/soap
```

Arraste agora para o formulário, da paleta *DataSnap*, o componente *ConnectionBroker (ConexaoB)* que abstrai para o *ClientDataSet* o tipo de conexão que será utilizada. Por exemplo, você pode trocar sua conexão de *SOAP* para *DCOM* ou *COM+* sem precisar alterar os *ClientDataSets*. Aponte sua propriedade *Connection* ao componente *ConexaoSOAP*.

Arraste agora para o *dmControle* três componentes *ClientDataSet (cdsProdutos, cdsFornecedores e cdsPedidosPorFornecedor)* da paleta *Data Access*. A ligação destes componentes será um pouco diferente as formas conhecidas em aplicações comuns *Win32*, onde ligávamos apenas ao *DataSetProvider*. Nesse caso está na camada *SOAP*. Para fazer esta conexão, aponte a propriedade *RemoteServer* de ambos os *ClientDataSets* para o componente *ConexaoB*, em seguida selecione-os de forma individual e relacione sua propriedade *DataSetProvider*, conforme esquema a seguir:

cdsProdutos -> dspProdutos;
cdsFornecedores -> dspFornecedores;
cdsPedidosPorFornecedor -> dspPedidosPorFornecedor.

Para ambos, com o botão direito do mouse, selecione a opção *FetchParams* a fim de carregar os parâmetros a serem passados para camada *SOAP* já nos *ClientDataSets*, realizando a configuração do *DataType* de cada um de acordo com o tipo de valor que deverá receber. Adicione ainda ao *Fields Editor* de cada *ClientDataSet* todos os *Fields (Add All Fields)* disponíveis na consulta *SQL* através da ligação com o *DataSetProvider* e a camada servidora.

Finalizada a configuração dos componentes de acesso a dados no *Data Module*, iniciaremos agora a construção dos formulários que possibilitarão a interatividade do usuário com o sistema. No menu *File>New>Form (frmCadFornecedores)* adicione um novo formulário. Altere sua propriedade *Caption* para *Cadastro de Fornecedores*. Vá agora até o *dmControle*, abra o *Fields Editor* do *cdsFornecedores*, retorne ao *frmCadFornecedores* e arraste todos os *Fields* para a tela, confirmando a mensagem da criação de um *DataSource (dsDados)* no formulário.

Veja que todos os campos da tabelas agora já estão disponíveis no formulário, bastando você organizá-los. Arraste também no topo do formulário o componente *DBNavigator* e aponte sua propriedade *DataSource* ao *dsDados*. Logo abaixo ao *DBNavigator*, adicione um *Button (btnConfirma)* alterando o *Caption* para *Confirmar Cadastro* e atribua o seguinte trecho de código ao evento *OnClick* do mesmo, responsável por persistir a gravação ou alteração de dados para a base de dados:

```
try
  dmControle.cdsFornecedores.ApplyUpdates(0);
except on E:Exception do
  showmessage('Problemas na gravação dos dados!' + #13 +
  E.Message);
end;
```

Arraste também ao formulário um componente *DBGrid*, ligue-o através da propriedade *DataSource* ao *dsDados* e com um duplo clique sobre o mesmo. No gerenciador de colunas adicione todas as colunas disponíveis usando a opção *Add>Aall fields*, onde serão listados as informações de fornecedores já cadastrados. Adicione ainda ao evento *OnClose* do formulário as seguintes linhas de código, garantindo que assim que sair do formulário o *DataSet* de fornecedores também será fechado através do *Close* no *ClientDataSet*.

```
try
  dmControle.cdsFornecedores.Close;
except
end;
```

Também ao evento *OnShow* da página, adicione o código da **Listagem 1**, responsável por abrir a conexão com o protocolo *SOAP* e em seguida atribua um valor ao parâmetro de busca do *ClientDataSet* Atribuiremos apenas o sinal % indicando que não haverá filtro e todos os registros serão exibidos. Por fim, abrimos o componente *ClientDataSet* e fechamos novamente a conexão com o protocolo *SOAP* que não tem necessidade de permanecer aberta, visando que todos os dados já foram carregados e permanecerão em *Pool* no sistema.

Listagem 1. OnShow do formulário de fornecedores

```
procedure TfrmCadFornecedores.FormShow(Sender: TObject);
begin
  try
    dmControle.ConexaoSOAP.Open;
    try
      dmControle.cdsFornecedores.Params[0].AsString := '%%';
      dmControle.cdsFornecedores.Open;
    finally
      dmControle.ConexaoSOAP.Close;
    end;
  except
    dmControle.cdsFornecedores.Close;
  end;
end;
```

Agora, vá até o formulário principal e adicione ao evento *OnClick* do menu fornecedores a linha de código a seguir. Adicione também a *Uses* do formulário de fornecedores ao formulário principal (vá em *File>Use Unit*). Após isso execute o sistema e veja o funcionamento do cadastro de fornecedores (**Figura 6**).

```
frmCadFornecedores.ShowModal;
```

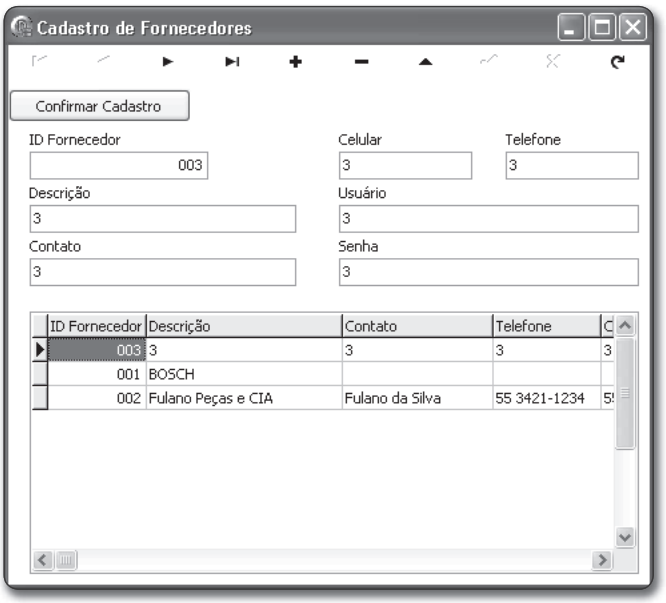


Figura 6. Cadastro de fornecedores

Iniciaremos agora a construção do formulário para cadastro de produtos. No menu *File>New>Form (frmCadProdutos)* adicione o formulário ao projeto, altere seu *Caption* para *Cadastro de Produtos*. Da mesma forma como fizemos com o cadastro de fornecedores, vá até o *dmControle* e abra o *Fields Editor* do *cdsProdutos* e retorne ao formulário arrastando todos os *Fields* para o mesmo, confirmando a mensagem da criação de *DataSource (dsDados)*. Arraste também ao topo um componente *DBNavigator* e aponte sua propriedade *DataSource* ao *dsDados*. Logo abaixo do *DBNavigator*, adicione um *Button (btnConfirma)* e altere seu *Caption* para *Confirmar Cadastro*, adicionando o código a seguir em seu evento *OnClick* com a função de persistir o cadastro e alterações realizadas na base de dados.

```
try
  dmControle.cdsProdutos.ApplyUpdates(0);
except on E:Exception do
  ShowMessage('Problemas na gravação dos dados!' + #13 +
  E.Message);
end;
```

Adicione também ao formulário um componente *DBGrid* e relacione-o através da propriedade *DataSource* ao *dsDados*. Com um duplo clique, adicione todas as colunas disponíveis provenientes dos *fields* do *cdsProdutos*. Nesse *grid* serão mostrados todos os registros de produtos já gravados. Para que os registros sejam carregados logo na abertura do formulário, é necessário que façamos a abertura do objeto *DataSet*, onde iremos atribuir ao evento *OnShow* do formulário o código da

Listagem 2, que fará a abertura da conexão com o protocolo *SOAP*, carregamento dos dados para a memória e em seguida o fechamento da conexão.

Listagem 2. OnShow do formulário de produtos

```
procedure TfrmCadProdutos.FormShow(Sender: TObject);
begin
  try
    dmControle.ConexaoB.Open;
    try
      dmControle.cdsProdutos.Params[0].AsString := '%%';
      dmControle.cdsProdutos.Open;
    finally
      dmControle.ConexaoB.Close;
    end;
  except on E:Exception do
    ShowMessage('Problemas na abertura da tabela!' + #13 +
      E.Message);
  end;
end;
```

Adicione também ao evento *OnClose* do formulário as seguintes linhas de código que serão responsáveis pelo fechamento do objeto *DataSet* no momento que o usuário sair do formulário.

```
try
  dmControle.cdsProdutos.Close;
except
end;
```

Retorne ao formulário principal e adicione agora ao evento *OnClick* do menu de produtos a seguinte linha de código, em seguida execute novamente a aplicação realizando o cadastro de alguns produtos (**Figura 7**).

```
frmCadProdutos.ShowModal;
```

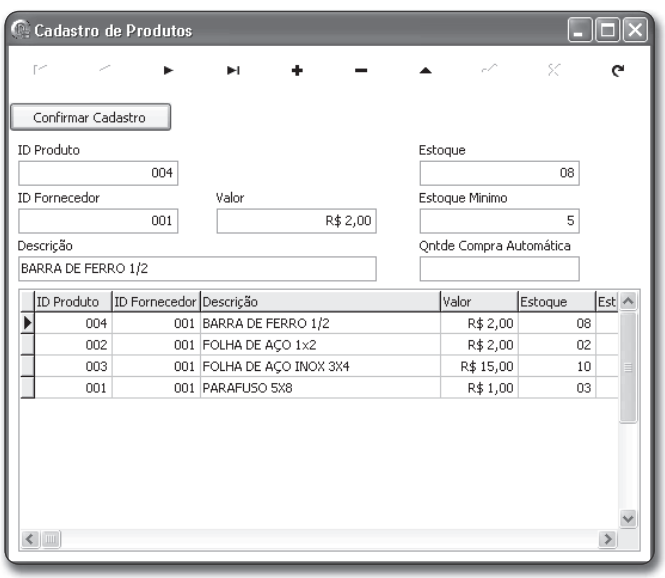


Figura 7. Cadastro de produtos

Para o processo de baixas de estoque dos produtos, a fim de que alcancem os limites mínimos de quantidades armazenadas, criaremos um simples formulário que exibirá os códigos de produtos e que quando selecionados será chamado um método da *Unit iFuncoes.pas* que fará a atualização da quantidade. Para adicionar o formulário, vá ao menu *File>New>Form (frmProdutosUtil)* para adicionar o formulário e em seguida altere seu *Caption* para *Utilização de Produtos*.

Arraste para o formulário quatro componentes *Label*, alterando o *Caption* de dois para *ID Produto* e de outros dois para *Quantidade Utilizada*, em seguida arraste dois componentes *Edit (edtProduto, edtQntde)* definindo o *Text* de ambos para *000*. Adicione também dois componentes *ListBox (ListBox1, ListBox2)* onde serão exibidos as informações de uma simulação de venda dando a baixa no estoque. Por último, adicione um *Button (btnGravar)* que será o responsável por fazer a comunicação com a camada *SOAP* dando a baixa no estoque. Organize os componentes no formulário de acordo com a **Figura 8.**

No processo de codificação, basta apenas que invoquemos o método *AtualizaEstoqueProduto* importado da *WSDL*. Nesse método passaremos como parâmetros o código do produto a ser alterado, a quantidade a ser modificada e a operação a ser executada, sendo neste caso "-", podendo utilizar o mesmo método em operações de reposição de estoque, alterando a ação para "+". Adicione ao evento *OnClick* do *btnGravar* o código da **Listagem 3** que fará a movimentação de produtos.

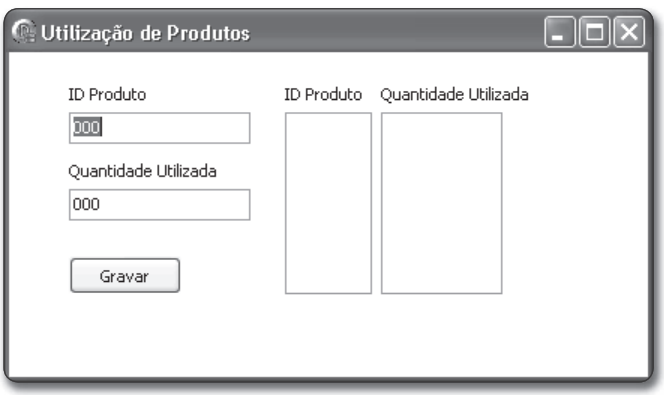


Figura 8. Cadastro de utilização de produtos

Listagem 3. Movimentação de produtos

```
procedure TfrmProdutosUtil.btnGravarClick(Sender: TObject);
begin
  try
    if GetIfuncoes().AtualizarEstoqueProduto(
      edtProduto.Text,edtQntde.Text,'-') then
    begin
      ListBox1.Items.add(edtProduto.Text);
      ListBox2.Items.add(edtQntde.Text);
      edtProduto.Text := '000';
      edtQntde.Text := '00';
       GetIfuncoes().MontaPedidos;
    end;
  except

  end;
end;
```

Observe nesse código da **Listagem 3** que estamos chamando os métodos disponíveis na *Unit iFuncoes.pas*, e através deste executamos a função de atualização de estoque dos produtos que, quando carregada com parâmetros vindos dos componentes da tela, irá retornar um valor *booleano* quando executada.

Fazendo com que os produtos adicionados sejam listados juntamente com a quantidade nos componentes *ListBox*. No formulário principal, vá até o menu de utilização de produtos e no seu evento *OnClick* adicione a seguinte linha de código, responsável por exibir o formulário ao clique sobre o menu:

```
frmProdutosUtil.ShowModal;
```

Ainda neste sistema do cliente, faremos um formulário onde o cliente poderá acompanhar os pedidos abertos para cada fornecedor. No menu File>New>Form (frmAcompPedidos) adicione um novo formulário alterando o Caption para Acompanhamento de Pedidos Ativos por Fornecedor. Arraste para o formulário dois componentes Label e altere o caption para "Pedidos Ativos por Fornecedor" e "ID do Fornecedor", adicione também um componente Edit (edtFornecedor) e um Button (btnBusca) que serão os componentes utilizados para realizar a pesquisa.

Para exibição dos resultados, adicione um componente DataSource (dsConsulta) e ligue-o ao cdsPedidosPorFornecedor que está no dmControle, sendo que será necessário adicionar a Unit do dmControle a este formulário. Adicione também um componente DBGrid para exibição dos registros, aponte sua propriedade DataSource para o dsConsulta e com um duplo clique sobre o mesmo, adicione todas as colunas de fields presentes no ClientDataSet. Para a codificação, será apenas necessário que o evento OnClick do botão de busca seja codificado, no qual deverá adicionar o código da Listagem 4 para a pesquisa.

Listagem 4. Pesquisa de pedidos por fornecedor

```
procedure TfrmAcompPedidos.btnBuscaClick(Sender: TObject);
begin
  try
    dmControle.ConexaoB.Open;
    try
      dmControle.cdsPedidosPorFornecedor.Params[0].
        AsString := edtFornecedor.Text;
      dmControle.cdsPedidosPorFornecedor.Open;
    finally
      dmControle.ConexaoB.Close;
    end;
  except
    showmessage('Não foi possível realizar a consulta!');
  end;
end;
```

Nesse bloco de códigos, estamos abrindo o componente de comunicação com o protocolo *SOAP* e em seguida, atribuímos um valor ao parâmetro de busca do *ClientDataSet*, parâmetro este que deverá receber o código de algum fornecedor. Por fim, abrimos a pesquisa trazendo todos os pedidos por fornecedor selecionado, deixando as informação em *pool* do *ClientDataSet* e fechamos o canal de comunicação com a camada *Server* em seguida.

Novamente no formulário principal, adicione ao menu acompanhamentos o seguinte código, responsável por exibir o formulário ao clique do menu, em seguida execute o sistema e consulte pedidos de fornecedores no formulário (**Figura 9**).

```
frmAcompPedidos.ShowModal;
```

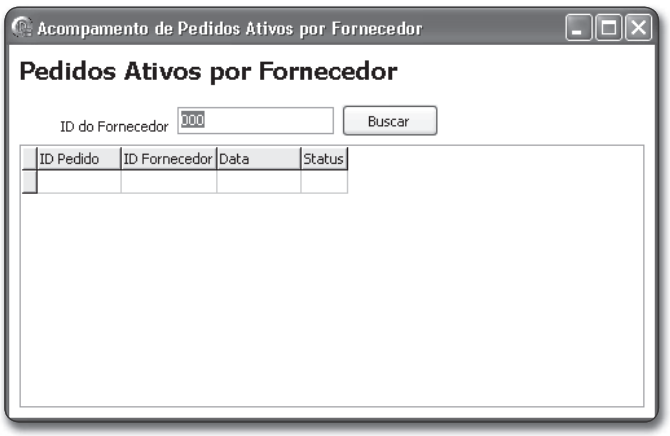


Figura 9. Consulta de pedidos por fornecedor

Para o último processo da aplicação cliente, adicionaremos logo ao centro do formulário principal um botão responsável por executar a rotina de pedidos. Essa rotina poderia ser programada para disparar ao abrir o sistema ou até mesmo um serviço rodando constantemente em algum servidor e que realizasse uma verificação de quantidades em estoque de tempos em tempos, fazendo os pedidos quando necessários. No *Button (btnMontaPedidos)* adicionado ao formulário, altere o *Caption* para *Montar Pedidos* e em seu evento *OnClick* atribua a seguinte linha de código:

```
ShowMessage(GetIfuncoes().MontaPedidos);
```

O código irá disparar a rotina *MontaPedidos* no lado do servidor (*SOAP)* verificando toda a lista de produtos com quantidade em limite para realizar pedidos, fazendo-os quando necessário. Ao final de todas as verificações de produtos e geração de pedidos, uma mensagem de sucesso na operação será exibida na tela. Para melhor entender, execute o sistema e faça as simulações de pedidos. Dessa forma, finalizamos a criação do sistema para o Cliente e daremos início ao desenvolvimento da camada cliente do fornecedor.

Criando a aplicação cliente do fornecedor

Quando falamos em *aplicação cliente do fornecedor*, nos referimos ao executável que será distribuído para os vários fornecedores desta empresa, sendo que cada um possuirá uma autenticação para acesso ao sistema, dando possibilidade de apenas visualizar seus próprios pedidos.

Criaremos um novo projeto para esta nova aplicação, no menu *File>New>VCL Forms Application – Delphi for Win32* crie uma nova aplicação, alterando inicialmente o nome do formulário principal para *frmLogin* e sua propriedade *Caption* para *Login - Identificação do Fornecedor*.

Semelhante ao exemplo anterior, também nesse usaremos de dois processos distintos de comunicação com a camada *SOAP*, um facilitando a chamada de métodos e outro para possibilitar uma fácil ligação entre componentes desta camada

cliente com componentes de ligação a dados da camada *Server*. Semelhante ao processo do exemplo anterior faça novamente a importação da *WSDL*, utilizando o endereço a seguir e observe a criação de uma nova *Unit* vinculada ao projeto (*ifuncoes.pas*):

```
http://localhost/Pedidos/PedidosSoapServer.dll/wsdl/Ifuncoes
```

Realizada a importação da *WSDL*, adicione a este projeto também um *Data Module* salvando sua *Unit* como *untControle.pas*. No *Data Module* faremos a utilização do endereço da *WSDL* através de componentes. Arraste para o formulário da paleta *WebServices* o componente *SoapConnection (ConexaoSOAP)* que fará a comunicação direta com o protocolo e adicione a sua propriedade *URL* o caminho de hospedagem do serviço *SOAP* através do seguinte endereço:

```
http://localhost/Pedidos/PedidosSoapServer.dll/soap
```

Arraste agora para o formulário, da paleta *Data Snap*, o componente *ConnectionBroker (ConexaoB)* e aponte sua propriedade *Connection* ao componente *ConexaoSOAP.* Arraste agora para o *dmControle* dois componentes *ClientDataSet (cdsPedidos e cdsItensPedidos)* da paleta *Data Access.* Na ligação destes componentes aponte a propriedade *RemoteServer* de ambos os *ClientDataSets* para o componente *ConexaoB*, em seguida selecione-os de forma individual e relacione sua propriedade *DataSetProvider*, conforme esquema a seguir:

cdsPedidos -> dspPedidosPorFornecedor;
cdsItensPedidos -> dspItens_Pedido.

No *cdsPedidos* clique com o botão direito do mouse, selecione a opção *FetchParams* a fim de carregar os parâmetros a serem passados para camada *SOAP* já nos *ClientDataSets*, realizando a configuração do *DataType* de acordo com o tipo de valor que deverá receber. Adicione ainda ao *Fields Editor* de cada *ClientDataSet* todos os *Fields (Add All Fields)* disponíveis na consulta *SQL* através da ligação com o *DataSetProvider* e a camada servidora.

Retornando ao formulário de *login*, arraste para a tela dois componentes *Label* e altere o *Caption* para *Usuário* e *Senha*, dois componente *Edit (edtUsuario, edtSenha)* e ainda dois *Button (btnOk, btnFecha)* e organize-os da forma como melhor se adaptarem. Ao evento *OnClick* do botão *btnFecha* atribua simplesmente *Close*, que fechará o formulário ao ser clicado. Já no botão *btnOk* adicione ao evento *OnClick* o código da **Listagem 5** que irá comparar o usuário e senha digitados aos cadastrados na base de dados. Declare também no método *public* da *Unit* as seguintes variáveis:

```
public
    { Public declarations }
    var
    NomeFornecedor, CodigoFornecedor: String;
  end;
```

Listagem 5. Login do fornecedor

```
procedure TfrmLogin.btnOkClick(Sender: TObject);
var
  c: WideString;
  n: WideString;
begin
  NomeFornecedor := '';
  CodigoFornecedor := '';
  n := '';
  c := '';
  if GetIfuncoes().IdentificaFornecedor(edtUsuario.
   Text,edtSenha.Text,c,n) then
    begin
      CodigoFornecedor := c;
      NomeFornecedor := n;
      frmLogin.Visible := false;
      frmPrincipal.showmodal;
    end
  else
    ShowMessage('Usuário ou senha inválidos!!!');
end;
```

No código de *login* após a criação das variáveis, estamos fazendo uma chamada ao método de identificação do fornecedor, passando-lhe como parâmetro o usuário e a senha digitados em tela, também duas variáveis que receberão valores após o resultado. Caso o retorno da função seja verdadeiro, autorizamos o *login* e chamamos o formulário principal. Em caso contrário, uma mensagem de erro é exibida.

Para a configuração do formulário principal, adicione um novo *form* através do menu *File>New>Form* alterando seu nome para *frmPrincipal* e o *Caption* para *Terminal Cliente - Controle Atomático de Pedidos*. Na parte superior do formulário adicione um componente *Panel* e dentro do mesmo adicione 5 componentes *Label (Label1, lblFornecedor, Label2, lblHora, lblCodigoFornecedor)* que serão utilizados para guardar informações referentes ao fornecedor ou como textos informativos.

Para o *Label1* altere o *Caption* para *Fornecedor:* e o *Label2* para *Tempo de Busca de Pedidos.* Adicione também um componente *Edit (edtSeg)* que receberá o intervalo de tempo para a pesquisa de novos pedidos e um componente *Button (btnGravaSeg)* para aplicar as alterações de tempo.

Logo abaixo do *Panel*, adicione dois componentes *GroupBox* e dentro de cada um destes adicione um componente *DBGrid*. Para relacionar os *DBGrids*, usaremos dois componentes *DataSource* que deverão ser adicionados ao formulário. Utilizando o menu *File>Use Unit* selecione a *Unit* do *Data Module* para que possamos relacionar com os componentes *ClientDataSet*. Selecione o *DataSource1* destinado a exibição de pedidos e ligue-o ao *cdsPedidos*, em seguida selecione o *DataSource2* destinado aos itens do pedido e ligue-o ao *cdsItensPedidos*.

Para relacionar os componentes *DBGrid*, selecione o *DBGrid1* e relacione-o na propriedade *DataSource* ao *DataSource1*, em seguida com um duplo clique sobre o mesmo, adicione todas as colunas ao *grid*. Repita a operação com o *DBGrid2* relacionando-o ao *DataSource2* e adicione também os *fields* ao *grid*. Para simular um relacionamento do tipo *master-detail*, onde ao trocar de pedido os itens se atualizem automaticamente, adicione ao evento *OnDataChange* do *DataSource1* o código da **Listagem 6** que fará uma verificação quanto a quantidade de registros, onde se houve algum, o código do pedido selecionado será repassado como parâmetro para a busca dos itens do mesmo.

Listagem 6. Simulando master-detail

```
procedure TfrmPrincipal.DataSource1DataChange(Sender: TObject;
Field: TField);
begin
  if DataSource1.DataSet.RecordCount > 0 then
  begin
    dmControle.cdsItensPedidos.close;
    dmControle.cdsItensPedidos.Params[0].AsInteger :=
     dmControle.cdsPedidosPED_CODIGO.AsInteger;
    dmControle.cdsItensPedidos.Open;
  end;
end;
```

Selecionando agora o formulário, faremos a codificação do evento *OnShow*, onde apenas iremos exibir nos componentes *Label* informações referentes ao fornecedor, como o nome e código do mesmo. Adicione as seguintes linhas de código ao evento:

```
lblFornecedor.Caption := frmLogin.NomeFornecedor;
lblCodigoFornecedor.Caption := frmLogin.CodigoFornecedor;
```

Para realizar as configurações de intervalo de tempo para as pesquisas, adicione um componente *Timer* à tela e no evento *OnTimer* adicione o código da **Listagem 7**. O código manterá atualizado o *label* com a hora da última atualização, além de abrir um canal de comunicação com a camada *SOAP*, fornecer um parâmetro com o código do fornecedor ao *ClientDataSet* de pedidos e verificar se há algum novo, fechando a conexão logo em seguida, exibindo os resultados na tela.

Listagem 7. Verificação de novos pedidos

```
procedure TfrmPrincipal.Timer1Timer(Sender: TObject);
begin
  lblHora.Caption := TimeToStr(Time);
  try
    dmControle.ConexaoB.Open;
    try
      dmControle.cdsPedidos.close;
      dmControle.cdsPedidos.Params[0].AsString := lblCodigoFornecedor.
       Caption;
      dmControle.cdsPedidos.Open;
    finally
      dmControle.ConexaoB.Close;
    end;
  except

  end;
end;
```

Para alterar o intervalo de tempo que o sistema deverá verificar a existência de novos pedidos, adicione ao clique do *btnGravaSeg* o código a seguir, que definirá um novo intervalo de tempo, ativando o inicio da contagem de tempo.

```
Timer1.Interval := StrToInt(edtSeg.Text)*1000;
Timer1.Enabled := True;
```

Para gerenciamento de pedidos, adicione a tela um componente *PopupMenu*, e com um duplo clique adicione a ele o menu *Pedido Encaminhado*, que será utilizado no momento em que o fornecedor encaminhou o pedido ao cliente. Ao evento *OnClick* do mesmo adicione a seguinte linha de código, que irá se conectar aos métodos da camada *SOAP* executando a função de alteração de status do pedido, recebendo como parâmetro o código do pedido e a sigla "E" indicando o *status* enviado.

```
GetIfuncoes().AlterarStatusPedido(dmControle.cdsPedidosPED_CODIGO.
asstring,'E');
```

Por último, adicione ainda ao evento *OnClose* do formulário a seguinte linha de código, responsável por fechar o *form* de *login* que permanece aberto durante a existência do formulário principal.

```
frmLogin.Close;
```

Compile e execute agora o sistema, utilizando como base de *login* informações que foram cadastradas durante a construção do primeiro exemplo no cadastro de fornecedores. Faça o gerenciamento de pedidos existentes ou crie novos a fim de testar a integração entre as duas aplicações *cliente* e a camada *SOAP servidora*.

Conclusão

Conhecemos neste artigo uma das formas possíveis de integração de sistemas em clientes e ambientes diferentes. São sistemas como estes que agilizam a comunicação e muitos processos que rodam em grandes empresas, como no exemplo do próprio artigo, uma empresa do setor metal-mecânico. Estas implementações podem ser feitas em outras situações de sistemas, onde você poderá implementar mais rotinas aperfeiçoando-as com itens de segurança e demais ferramentas disponíveis. Abraço e até o próximo artigo. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Hospede seus sites com os profissionais da Hospedix.

3 DOMÍNIOS PRINCIPAIS

200 DOMÍNIOS HOSPEDADOS

100GB DE TRANSFERÊNCIA MENSAL

30GB PARA E-MAILS

2GB DE ESPAÇO EM DISCO

Clientes Hospedix ainda podem optar por uma de nossas promoções:

1 registro de domínio (.com .net .org ou .info) grátis e isento de taxas anuais.

ou

1 registro de domínio (.com.br) por apenas R$15,00 anuais enquanto for cliente.

Tudo isso por apenas **R$ 25,90** por mês

Assine agora mesmo e descubra por que Hospedix é a hospedagem de sites preferida entre os profissionais.

HOSPEDIX
HOSPEDAGEM PROFISSIONAL
www.hospedix.com.br

30 DIAS GRÁTIS

Ao assinar, digite o código de desconto exclusivo para leitores desta revista e ganhe 30 dias inteiramente grátis.

RVSTMEDIA

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET

Chaves com ClientDataSet

Geração de chaves com master-detail e ClientDataSet

Manoel Edesio B. da Silva
manoeledesio@edesoft.com.br
Detém as certificações em Borland C++ Builder, Borland Delphi for Win32 e Borland Delphi for .NET. Palestrante na Borland Conference de 2004, 2005, 2006 e 2007. Atualmente é sócio/diretor da EdeSoft, empresa especializada em consultoria, treinamentos customizados, mentoring, outsourcing e desenvolvimento de software customizados. A EdeSoft também atua atualmente em projetos especializados de mobilidade com utilização de celulares, PDAs e tem auxiliado as empresas de software a melhorar a produtivdade do seu desenvolvimento e reduzir custos com manutenção

Resumo do DevMan

Uma das maiores dificuldades dos desenvolvedores é sem dúvida quando temos dependências de chaves em um relacionamento master-detail do tipo Pedido x ItensPedido. Nesses casos precisamos passar para a tabela de detalhe a chave primária da tabela master, porém essa chave é gerada através de uma trigger (gatilho) que está no banco de dados. Isso cria uma dificuldade aos desenvolvedores para se ter controle sobre o processo, pois parte dele fica no banco de dados. Sendo assim se acaba optando por outras técnicas mais trabalhosas que podem deixar a aplicação com falhas no processo.

Neste artigo iremos criar toda uma estrutura para se trabalhar um relacionamento master-detail onde as chaves são geradas pelo banco de dados e teremos o controle dessa geração através de uma aplicação multicamadas com a utilização dos componentes DataSetProvider e ClientDataSet.

Nesse artigo veremos

- Geração de tabelas, generators e Stored Procedure em banco de dados Firebird;
- Criação de servidor de aplicação;
- Geração das chaves através de eventos do DataSetProvider;
- Criação da aplicação cliente e conexão através de DataSnap;
- Configuração do ClientDataset para gravação dos dados.

Qual a finalidade

Definir uma estrutura para se trabalhar com relacionamentos master-detail através do ClientDataSet onde as chaves primárias são geradas automaticamente pelo banco de dados.

Quais situações utilizam esse recurso?

Relacionamentos master-detail onde as chaves são geradas pelo banco de dados através de generators.

Todos nós sabemos ou pelo menos deveríamos saber que boa parte de um sistema baseia-se em *master-detail*. Para os iniciantes, *master-detail* significa *mestre-detalhe*, ou seja, tabelas que dependem de outras tabelas para fazerem sentido. Alguns exemplos são clássicos tais como *Notas Fiscais* e *Itens da Nota Fiscal*. Evidentemente que em um sistema emissor de *Notas Fiscais* precisaremos de duas tabelas, como mencionado a pouco. A primeira tabela conterá o cabeçalho da nota. São os dados fixos, essenciais como, por exemplo, *Razão Social*, *Endereço*, *Valor Total da Nota*, *CNPJ do Cliente*, *CNPJ do Fornecedor*, entre outros. A segunda tabela refere-se aos itens dessa nota, que são variáveis e podem ser mais de um. Por isso dizemos que seu relacionamento é *1:n*, ou, 1 para muitos. Em uma nota fiscal podemos vender mais de um item, por isso 1 para muitos.

Para os mais experientes, isso tudo não é novidade. O que muitos, até mesmo experientes, sentem dificuldade no dia-a-dia é o controle das chaves que fazem a ligação entre ambas tabelas. Os exemplos que dei aqui são apenas entre duas tabelas, mas imaginem diversas tabelas interligadas. Como controlar tantas chaves?

A idéia desse artigo é sanar as principais dúvidas referentes ao assunto e demonstrar técnicas para utilização de chaves com *ClientDataSet*.

Para isso iniciaremos o nosso exemplo construindo um banco de dados com uma estrutura para que se possa simular a situação na qual o nosso exemplo propõe. Em nosso exemplo iremos utilizar o *Firebird* como banco de dados.

Nota: Não entrarei em detalhes quanto ao download e instalação do banco de dados por não fazer parte do escopo desse artigo, porém a dica é efetuar o download do *Firebird* no site do desenvolvedor em *www.firebirdsql.org*.

Criando o banco de dados

Como mencionado vamos criar um *BD* no *Firebird*. Para isso, basta digitar o script da **Listagem 1** no seu programa preferido para rodar scripts e executar. Em nosso script criamos um banco utilizando o *Dialect 3*, *Page Size 8192* e o *Character Set* como nenhum (*None*). No endereço do banco colocamos *'127.0.0.1:<caminho>DBExemplo.fdb'*, onde *127.0.0.1* é o *host* (*servidor*) do banco, *<caminho>* deve ser trocado pela letra de drive e diretório onde deseja salvar o arquivo do *BD* e por fim o nome desse *BD*. Além dessas informações também é possível colocar a porta do servidor junto ao *host*, por exemplo: *127.0.0.1/3050*.

Listagem 1. Script de criação do banco de dados

```
SET SQL DIALECT 3;
CREATE DATABASE '127.0.0.1:<Caminho>DBExemplo.fdb'
USER 'SYSDBA'
PAGE_SIZE 8192
DEFAULT CHARACTER SET NONE;
```

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora mesmo o portal do assinante ClubeDelphi e veja uma vídeo aula de Luciano Pimenta que mostra como utilizar o IBExpert para administração avançada de banco de dados Firebird/Interbase.

http://www.devmedia.com.br/articles/viewcomp.asp?comp=3082

Enfim, troque a string *'127.0.0.1:<Caminho>DBEXemplo.gdb'* pelo caminho e nome de banco de dados que desejar. Em seguida criaremos as tabelas. São elas: *TB_CLIENTE*, *TB_VENDEDOR*, *TB_PRODUTO*, e *TB_ITEMPEDIDO*. O objetivo básico do exemplo será efetuar a inclusão de um pedido onde teremos itens de pedido vinculados e também onde a chave primária do pedido será gerada pelo banco de dados em uma *trigger*. Porém a tabela *TB_ITEMPEDIDO* dependerá da chave primária gerada pela tabela *master* para a gravação dos dados. Na **Listagem 2** temos o *script* para criação das tabelas, chaves primárias, chaves secundárias e *generators*.

Listagem 2. Script de criação das tabelas de banco de dados

```
01 CREATE TABLE TB_CLIENTE (
02     ID_CLIENTE  INTEGER NOT NULL,
03     NM_CLIENTE  VARCHAR(50)
04 );
05
06 CREATE TABLE TB_ITEM_PEDIDO (
07     ID_PEDIDO   INTEGER NOT NULL,
08     ID_ITEM     INTEGER NOT NULL,
09     QTD         INTEGER,
10     PRECO_UNIT  FLOAT
11 );
12
13 CREATE TABLE TB_PEDIDO (
14     ID_PEDIDO    INTEGER NOT NULL,
15     ID_CLIENTE   INTEGER,
16     ID_VENDEDOR  INTEGER,
17     DT_EMISSAO   DATE,
18     VALOR_TOTAL  FLOAT,
19     VALOR_DESC   FLOAT
20 );
21
22 CREATE TABLE TB_PRODUTO (
23     ID_PRODUTO  INTEGER NOT NULL,
24     NM_PRODUTO  VARCHAR(30)
25 );
26
27 CREATE TABLE TB_VENDEDOR (
28     ID_VENDEDOR  INTEGER NOT NULL,
29     NM_VENDEDOR  VARCHAR(20)
30 );
31
32 ALTER TABLE TB_CLIENTE ADD CONSTRAINT PK_TB_CLIENTE PRIMARY KEY
33 (ID_CLIENTE);
34 ALTER TABLE TB_ITEM_PEDIDO ADD CONSTRAINT PK_TB_ITEM_PEDIDO
    PRIMARY KEY
35 (ID_PEDIDO, ID_ITEM);
36 ALTER TABLE TB_PEDIDO ADD CONSTRAINT PK_TB_PEDIDO PRIMARY KEY
37 (ID_PEDIDO);
38 ALTER TABLE TB_PRODUTO ADD CONSTRAINT PK_TB_PRODUTO PRIMARY KEY
39 (ID_PRODUTO);
40 ALTER TABLE TB_VENDEDOR ADD CONSTRAINT PK_TB_VENDEDOR PRIMARY KEY
41 (ID_VENDEDOR);
42
43 ALTER TABLE TB_ITEM_PEDIDO ADD CONSTRAINT FK_TB_ITEM_PEDIDO
    FOREIGN KEY
44 (ID_PEDIDO) REFERENCES TB_PEDIDO (ID_PEDIDO);
45 ALTER TABLE TB_ITEM_PEDIDO ADD CONSTRAINT FK_TB_ITEM_PEDIDO_
    ITEM FOREIGN
46 KEY (ID_ITEM) REFERENCES TB_PRODUTO (ID_PRODUTO);
47
48 CREATE GENERATOR GEN_TB_CLIENTE_ID;
49 CREATE GENERATOR GEN_TB_PEDIDO_ID;
50 CREATE GENERATOR GEN_TB_VENDEDOR_ID;
```

Definindo o método a ser utilizado

Em nosso exemplo a tabela *TB_PEDIDO* terá a sua chave gerada por meio do banco de dados através de uma *trigger* e o valor da chave armazenado em um *generator*. Porém em um relacionamento *master-detail* o detalhe depende da chave gerada pela tabela *master*, isso implica que precisamos controlar manualmente a geração da chave para que nossa aplicação possa passá-la à tabela detalhe na gravação dos dados. Para tal funcionalidade iremos criar uma *Stored Procedure* onde poderemos controlar a geração da chave que será armazenada por um *generator*. Além disso, iremos configurar uma *trigger* para que a mesma execute a *Stored Procedure* no caso de uma gravação ser feita diretamente pelo banco de dados para que a chave, ou simplesmente *ID*, possa ser gerado. O *script* de criação da *trigger, stored procedure* que pertencem à tabela *TB_PEDIDO* pode ser conferido na **Listagem 3**, também nessa listagem podemos encontrar as *triggers* das outras tabelas que utilizaremos em nosso exemplo.

Listagem 3. Script de criação das triggers e procedures

```
01 SET TERM ^ ;
02
03 CREATE PROCEDURE SP_GEN_PEDIDOID (
04     INCREMENTO INTEGER)
05 RETURNS (
06     NEWID INTEGER)
07 AS
08 BEGIN
09   EXIT;
10 END^
11
12 CREATE TRIGGER TB_CLIENTE_BI FOR TB_CLIENTE
13 ACTIVE BEFORE INSERT POSITION 0
14 AS
15 BEGIN
16   IF (NEW.ID_CLIENTE IS NULL) THEN
17     NEW.ID_CLIENTE = GEN_ID(GEN_TB_CLIENTE_ID,1);
18 END
19 ^
20
21 CREATE TRIGGER TB_PEDIDO_BI FOR TB_PEDIDO
22 ACTIVE BEFORE INSERT POSITION 0
23 AS
24    declare variable GEN_VAL INTEGER;
25 BEGIN
26  if ((new.ID_PEDIDO is null) or (new.ID_PEDIDO < 1)) then
27  begin
28     execute procedure sp_gen_pedidoid (1) returning_
       values(:Gen_Val);
29     new.ID_PEDIDO = Gen_Val;
30  end
31 END
32 ^
33
34 CREATE TRIGGER TB_VENDEDOR_BI FOR TB_VENDEDOR
35 ACTIVE BEFORE INSERT POSITION 0
36 AS
37 BEGIN
38   IF (NEW.ID_VENDEDOR IS NULL) THEN
39     NEW.ID_VENDEDOR = GEN_ID(GEN_TB_VENDEDOR_ID,1);
40 END
41 ^
42
43 SET TERM ^ ;
44
45 ALTER PROCEDURE SP_GEN_PEDIDOID (
46    INCREMENTO INTEGER)
47 RETURNS (
48     NEWID INTEGER)
49 AS
50 begin
51   NEWID = GEN_ID(gen_tb_pedido_id, INCREMENTO);
52   suspend;
53 end
54 ^
55
56 SET TERM ; ^
```

Construindo a aplicação no RAD Studio

O exemplo que construiremos pode ser desenvolvido utilizando qualquer versão do *Delphi* que possua o *ClientDataSet*. Nesse caso utilizaremos a versão *RAD Studio 2007* para criar uma nova aplicação *Win32* acessando o menu *File>New>VCL Forms Application – Delphi for Win32*. Ao criar o projeto em banco adicione um *TLabel* no formulário e altere a propriedade *Caption* para *Servidor de Aplicação.* Seguindo o nosso exemplo vamos salvar o projeto com o nome de *ServidorChaves* e a *Unit* do formulário com o nome de *ufrmServidor.pas* na pasta *C:\meuexemplo*, que é a mesma pasta onde foi salvo o nosso banco de dados. Na seqüência vamos acessar o menu *File>New>Other* e no item *Multitier* escolha *RemoteDataModule*, configurando o mesmo conforme **Figura 1** e após configurá-lo clique em *Ok.*

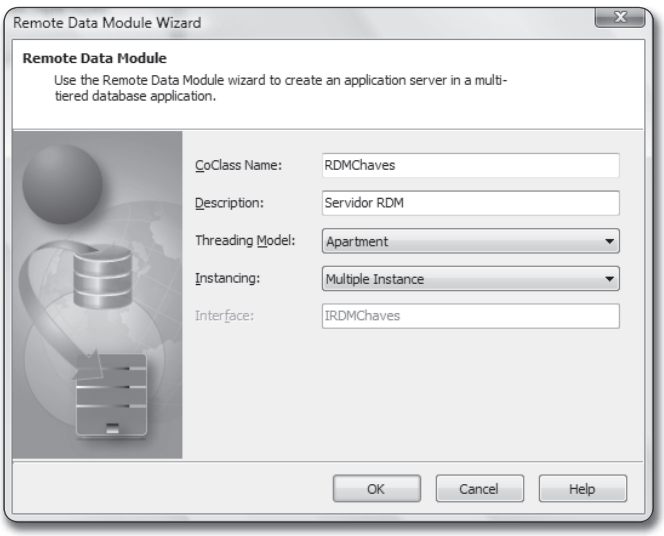


Figura 1. Configurando o servidor de aplicação

Em seguida vamos acessar o menu *File* e clique na opção *Save All* e salve a *Unit* com o nome *uRdmServidor.pas*. No nosso *RemoteDataModule* vamos agora adicionar um componente *SqlConnection* e vamos mudar o seu nome para *sqlConexao* em seguida na propriedade *ConnectionName* escolha o item *IBConnection* e configure a propriedade *Database* com o caminho do banco de dados criado que é *localhost:c:\meuexemplo\DBExemplo.fdb* e em seguida, adicione um *SQLDataSet*, um *DataSetProvider* e altere os seus nomes para *sqlProdutos* e *dspProdutos*. Depois configure a propriedade *SQLConnection* do *SqlDataSet* para o *sqlConexao* e na propriedade *CommandText* vamos colocar a seguinte sentença *SQL*:

```
SELECT * FROM TB_PRODUTO
```

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora mesmo o portal do assinante ClubeDelphi e veja uma vídeo aula de Everson Borges Volaco que mostra como utilizar o IBExpert para criação de Triggers e Generators.

http://www.devmedia.com.br/articles/viewcomp.asp?comp=4858

Já no componente *DataSetProvider* configure a propriedade *Dataset* para o *SqlDataSet*. Continuando vamos adicionar dois *SqlDataSets* e dois *DataSetProviders* e configurar os componentes mudando os seus nomes para *SqlClientes*, *SqlVendedores*, *DspClientes*, *DspVendedores* e na propriedade *CommandText* do *SqlClientes*:

```
SELECT * FROM TB_CLIENTE
```

No *SqlVendedores* incluímos na propriedade *CommandText*:

```
SELECT * FROM TB_VENDEDOR
```

Dando seqüência, vamos configurar o nosso relacionamento *master-detail*. Primeiramente vamos adicionar um *SqlDataSet* e mudar o nome para *sqlPedidos* configure também a propriedade *SqlConnection* para *sqlConexao* e na propriedade *CommandText* adicione a seguinte instrução *SQL*:

```
SELECT * FROM TB_PEDIDO
```

Em seguida adicione um *DataSetProvider* e mude o nome para *dspPedidos* e configure a propriedade *Dataset* para o *sqlPedidos*. Agora vamos configurar o detalhe do nosso relacionamento adicionamento um novo *sqlDataSet* e mudando seu nome para *sqlItemPedido*. Configure a propriedade *sqlConnection* para *sqlConexao* e na propriedade *CommandText* vamos colocar o *SQL*:

```
SELECT * FROM TB_ITEM_PEDIDO WHERE ID_PEDIDO=:ID_PEDIDO
```

Através desse *SQL* estamos criando um relacionamento entre a tabela *master* e a tabela *detalhe*. Em seguida vamos acessar a propriedade *Params* do componente e configurar as propriedades *DataType* e *ParamType* conforme a **Figura 2.**

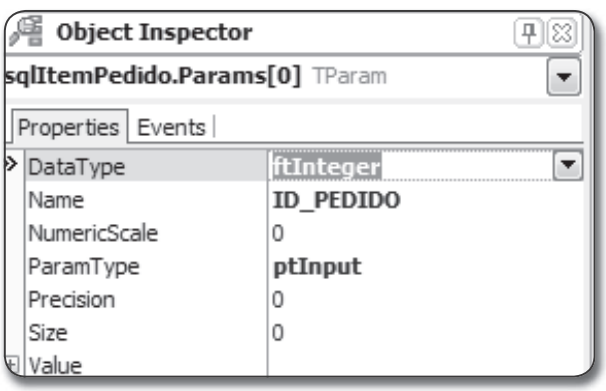


Figura 2. Configurando o parametro no sqlItemPedido

Finalizando a configuração do relacionamento vamos adicionar um *DataSource* e mudar o seu nome para *dsRelItemPedido*, apontar em seguida a propriedade *DataSet* do componente para *sqlPedidos*. Depois selecionar o componente *sqlItemPedido* e na propriedade *DataSource* configurar para o *dsRelItemPedido*.

Perceba que todo o processo descrito anteriormente parece complexo, mas não é. Esses são os passos essenciais e básicos para se configurar uma conexão via *dbExpress* em Delphi. Com o passar do tempo isso acaba se tornando rotina e até robotizado. O importante é entender bem o que é necessário para se configurar corretamente os componentes e com isso ganhar em produtividade e performance.

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora mesmo o portal do assinante ClubeDelphi e assista a primeira vídeo aula do mini-curso, composto de 20 vídeos, ministrado por Guinther Pauli que explora o uso do componente ClientDataSet.

http://www.devmedia.com.br/articles/viewcomp.asp?comp=5439

Trabalhando com eventos do provider

Vamos montar a estrutura para fazer com que na inclusão de um pedido e de seus itens de pedido, a chave primária gerada seja repassada de forma correta aos itens de pedido que estão envolvidos na inclusão. Continuando vamos adicionar um *sqlDataSet* e mudar o seu nome para *sqlProcIdPedido*, vamos configurar a propriedade *sqlConnection* para o *sqlConexao*, na propriedade *CommandType* escolha a opção *ctStoredProc* e na propriedade *CommandText* teremos uma lista com as *Stored Procedures* criadas em nosso banco de dados. Nessa lista aparecerá o nome *SP_GEN_PEDIDOID*, então é só selecioná-la e pronto.

Quando utilizamos um *ClientDataSet* gravamos os seus dados utilizando o método *ApplyUpdates*. Por sua vez, o *ApplyUpdates* dispara diversos eventos no *DataSetProvider*, que em nosso exemplo está no *Servidor de Aplicação*. Através desses eventos podemos programar diversas regras de negócios para simplificar as regras de processos do nosso software. Agora vamos selecionar o nosso componente *dspPedidos* e vamos abrir o evento *BeforeUpdateRecord* dando um duplo clique nele para adicionar o código do nosso tratamento. Nesse evento como temos duas tabelas relacionadas o método *ApplyUpdates* é apenas chamado na tabela *master* e com isso o evento é disparado a cada registro e nele passará os dados da tabela *master* e também os dados da tabela detalhe. Como faremos então para resolver o problema das chaves? É bem simples, veja **Listagem 4.**

Listagem 4. Implementando o evento BeforeUpdateRecord no DataSetProvider

```
01 procedure TRDMChaves.dspPedidosBeforeUpdateRecord(Sender: TObject;
02   SourceDS: TDataSet; DeltaDS: TCustomClientDataSet; UpdateKind:
03   TUpdateKind;
04 var Applied: Boolean);
05 begin
06   if (SourceDS = sqlPedidos) and
07      (DeltaDS.FieldByName('ID_PEDIDO').NewValue < 0) and
08      (UpdateKind = ukInsert) then
09   begin
10     sqlProcIdPedido.Close;
11     sqlProcIdPedido.Params[0].AsInteger := 1;
12     sqlProcIdPedido.ExecSQL;
13     DeltaDS.F ieldByName('ID_PEDIDO').NewValue :=
14          sqlProcIdPedido.Params[1].AsInteger;
15   end;
16
17   if DeltaDS.FindField('ID_PEDIDO') <> nil then
18      if UpdateKind = ukInsert then
19      begin
20        if (DeltaDS.FieldByName('ID_PEDIDO').NewValue < 0) then
21            DeltaDS.FieldByName('ID_PEDIDO').NewValue :=
22                sqlProcIdPedido.Params[1].AsInteger;
23      end;
24 end;
```

Estamos verificando inicialmente se o componente que está passando pelo evento é o *sqlPedidos* e se o valor do campo *IDPEDIDO* é negativo.

Nota: Na aplicação cliente faremos um tratamento para geração de chaves utilizando números negativos.

Seguindo verificamos se a ação é inclusão. Se todas as opções são verdadeiras então executamos a nossa *StoredProcedure* que irá gerar um novo *id* através do banco de dados e o passamos para o campo *ID_PEDIDO* do registro que está sendo gravado naquele momento. Continuando no nosso código fazemos o tratamento para verificar se existe um campo chamado *ID_PEDIDO*, se a ação que está sendo executada é uma inclusão e se o seu valor é negativo e caso seja verdadeira essas validações apenas passamos para o mesmo o valor já gerado pela *StoredProcedure*, assim garantindo aos itens de detalhe a atribuição da chave primária daquele pedido que representa o registro *master* para os seus detalhes.

Para concluir o servidor de aplicação deveremos registrar o mesmo em no sistema operacional. Fazemos isso acessando o menu *Run>Parameters* e no item *Parameters* vamos colocar */regserver* conforme **Figura 3.**

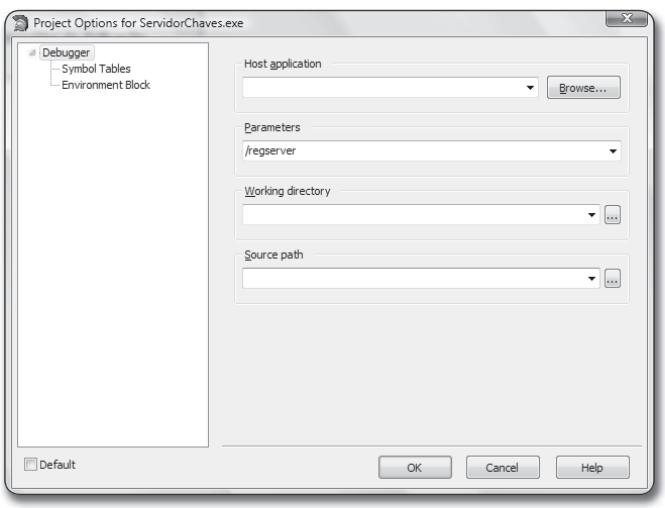


Figura 3. Registrando o servidor de aplicação no Windows

Depois de configurado vamos executar a aplicação pressionando *Ctrl+Shift+F9*. Executada a aplicação já podemos fechar a mesma, pois o projeto já foi registrado no *Windows*.

Criando a aplicação cliente

Criamos uma pequena aplicação servidora, agora o que precisamos é criar a versão cliente que fará o envio das informações ao servidor. Vamos criar uma nova aplicação no *RAD Studio 2007* acessando o menu *File>New>VCL Forms Application – Delphi for Win32*, em seguida vamos salvar o projeto na pasta `C:\MEUEXEMPLO\CLIENTE` com o nome de *ClienteChaves* e a *Unit* com o nome de *ufrmCliente.pas*.

Continuando vamos adicionar na aplicação um *DataModule*, através do menu *File>New>Other* e no item *Delphi Files* escolhendo a opção *DataModule*. Salve ele pressionando *Ctrl+S* e coloque o nome de *uDmCliente.pas*, além disso, vamos mudar sua propriedade *Name* para *DMCliente*. Para estabelecer a conexão com a aplicação servidora vamos adicionar ao *DataModule* um componente *SocketConnection*.

Nota: Quando utilizamos o componente *SocketConnection* para conseguirmos estabelecer a conexão com a aplicação servidora precisamos antes executar o aplicativo *scktsrvr.exe* que se encontra na pasta *bin* de instalação do RAD Studio.

Na propriedade *Address* do *SocketConnection* vamos colocar *127.0.0.1*, que é o endereço *IP* para máquina local. Na propriedade *ServerName* selecione o item *ServidorChaves.RDMChaves*.

Em seguida vamos adicionar três *ClientDataSets* e trocar o nome deles para: *CdsClientes*, *CdsVendedores* e *CdsProdutos*. Com o *CdsClientes* selecionado vamos configurar a propriedade *RemoteServer* para o *SocketConnection1* e na propriedade *ProviderName* vamos escolher o *provider* equivalente que é o *dspClientes*. Para concluir a configuração vamos repetir os passos para os componentes seguintes vinculando os mesmos aos *providers* equivalentes.

Para configurar os pedidos vamos adicionar um *ClientDataSet* ao nosso *DataModule* e trocar o seu nome para *cdsPedidos*, configurar a propriedade *RemoteServer* para o *SocketConnection1* e a propriedade *ProviderName* para o *dspPedidos*. Na seqüência para recuperar a estrutura da tabela de detalhe *ItemsPedido* vamos dar um duplo clique no *cdsPedidos* e pressionar *Ctrl+F* para adicionar todos os campos em sua estrutura. Como último campo deveremos ter o *sqlItemPedido* conforme **Figura 4.**

Figura 4. DatasetField do ItemPedido no ClientDataSet cdsPedidos

Adicionaremos agora mais um *ClientDataSet* e o chamaremos de *cdsItemsPedido*. Vamos configurar a sua propriedade

DataSetField para *cdsPedidossqlItemPedido*. Em seguida adicionaremos dois componentes *DataSource* e mudaremos o seus nomes para *dsPedidos* e *dsItemsPedido*. Vamos configurar a propriedade *Dataset* do *dsPedidos* para *cdsPedidos* e o *dsItemsPedido* para *cdsItemsPedido*.

Continuando as nossas configurações vamos declarar uma variável na seção *Private* chamada *FIDPedido*. Essa variável terá o papel de controlar os *IDs* gerados pela aplicação cliente para não ocorrer erro de chaves duplicadas na inclusão de novos pedidos. Vamos selecionar o evento *AfterOpen* do *cdsPedidos* e dar um duplo clique para abrir o seu evento. Aberto o evento vamos adicionar o seguinte código que irá inicializar a variável criada com o valor *0*.

```
FIDPEdido := 0;
```

Inicializada a variável vamos dar um duplo clique no evento *OnNewRecord* e colocar o código a seguir:

```
Dec(FIDPEdido);
DataSet.FieldByName('IDPEDIDO').AsInteger := FIDPEdido;
```

Através desse código estamos, a cada novo registro adicionado ao *cdsPedidos*, decrementando o valor da variável e assim gerando um valor de chave temporário e negativo. Geramos um número negativo para que não ocorra duplicidade de forma alguma de chaves com os dados que poderão estar no *cache* do *ClientDataSet*.

Finalizando as configurações dessa etapa, vamos selecionar o *cdsItemsPedido*, dar um duplo clique no evento *OnNewRecord* e adicionar o seguinte código:

```
DataSet.FieldByName('ID_PEDIDO').AsInteger :=
    cdsPedidos.FieldByName('ID_PEDIDO').AsInteger;
```

Esse código garante que ao se incluir um novo registro na tabela de detalhe estamos utilizando a chave temporária gerada pela tabela *master* do registro selecionado.

Configurando os campos de Lookup

Em nosso exemplo estamos utilizamos três tabelas com informações adicionais. São elas: *TB_CLIENTE*, *TB_PRODUTO* e *TB_VENDEDOR*. Como a idéia principal de nosso artigo é o tratamento das chaves não vamos sofisticar a aplicação no ponto de cadastro e consultas para essas tabelas. Sendo assim vamos utilizar campos *lookup* no *cdsPedidos* e no *cdsItemsPedido* para facilitar o nosso trabalho e agilizar as consultas desses items.

Vamos então dar um duplo clique no *cdsPedidos* e no editor de campos vamos clicar com o botão direito do mouse e escolher a opção *New Field*. Nessa opção criaremos um campo *lookup* para representar o *Cliente*, de acordo com as configurações **Figura 5.**

Em seguida vamos adicionar um segundo campo *lookup* chamado *Vendedor* e configurá-lo de acordo com a **Figura 6**.

Continuando o nosso exemplo vamos dar um duplo clique no *cdsItemsPedido* e pressionar *Ctrl+F* para adicionarmos todos os campos. Em seguida com o botão direito escolher a opção *New Field* e agora vamos criar um novo campo de *lookup* chamado *Produto* configurando-o conforme **Figura 7.**

Figura 5. Configurando o campo lookup no CdsPedidos chamado Cliente

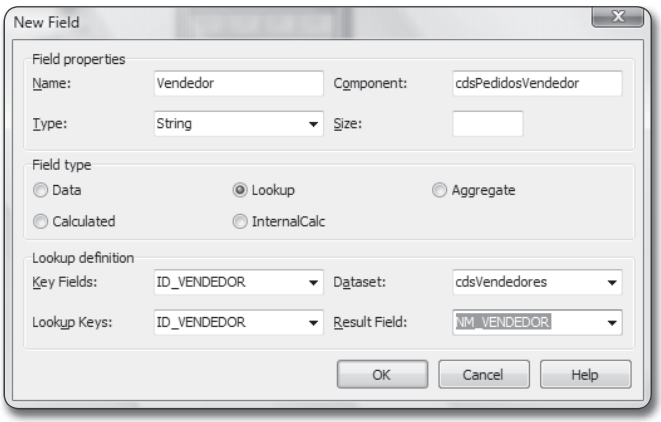


Figura 6. Configurando o campo lookup no CdsPedidos chamado Vendedor

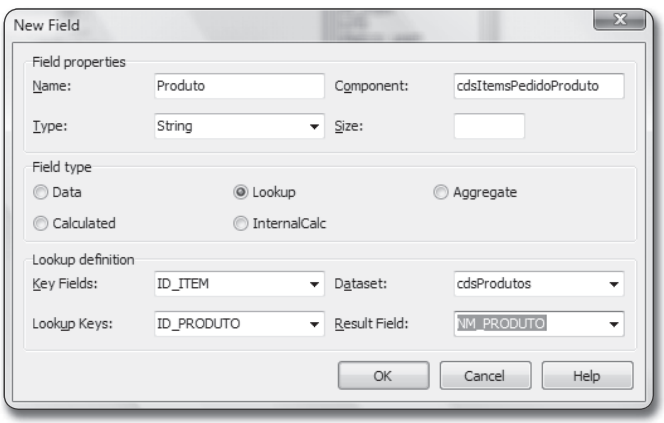


Figura 7. Configurando o campo lookup no CdsItemsPedido chamado Produto

Adicionando informação ao banco de dados

Antes de iniciarmos a criação da interface cliente, temos que preencher as tabelas que iremos utilizar para consulta em nossa aplicação. Precisaremos então preencher as tabelas *TB_PRODUTO*, *TB_CLIENTE* e *TB_VENDEDOR*. Então abra o seu gerenciador de banco de dados e execute as instruções *SQL* da **Listagem 5.**

Listagem 5. Adicionando informações ao Banco de Dados

```
01 INSERT INTO TB_CLIENTE (ID_CLIENTE, NM_CLIENTE) VALUES (1,
    'MANOEL
02 EDESIO');
03 INSERT INTO TB_CLIENTE (ID_CLIENTE, NM_CLIENTE) VALUES (2, 'ANA
04 BEATRIZ');
05 INSERT INTO TB_CLIENTE (ID_CLIENTE, NM_CLIENTE) VALUES (3, 'MIRIAN
06 CARDINOT');
07 INSERT INTO TB_CLIENTE (ID_CLIENTE, NM_CLIENTE) VALUES (4, 'PEDRO
08 HENRIQUE');
09 INSERT INTO TB_CLIENTE (ID_CLIENTE, NM_CLIENTE) VALUES (5, 'JOÃO
10 BATISTA');
11
12 INSERT INTO TB_PRODUTO (ID_PRODUTO, NM_PRODUTO) VALUES (1,
    'RAD STUDIO');
13 INSERT INTO TB_PRODUTO (ID_PRODUTO, NM_PRODUTO) VALUES (2,
    'STARTEAM');
14 INSERT INTO TB_PRODUTO (ID_PRODUTO, NM_PRODUTO) VALUES (3,
    'CALIBER');
15 INSERT INTO TB_PRODUTO (ID_PRODUTO, NM_PRODUTO) VALUES (4,
    'INTERBASE');
16 INSERT INTO TB_PRODUTO (ID_PRODUTO, NM_PRODUTO) VALUES (5,
    'JBUILDER');
17
18 INSERT INTO TB_VENDEDOR (ID_VENDEDOR, NM_VENDEDOR) VALUES (1,
    'CARLOS
19 GUEDES');
20 INSERT INTO TB_VENDEDOR (ID_VENDEDOR, NM_VENDEDOR) VALUES (2,
    'MARCOS
21 SILVA');
22 INSERT INTO TB_VENDEDOR (ID_VENDEDOR, NM_VENDEDOR) VALUES (3,
    'JOÃO
23 GOMES');
24 INSERT INTO TB_VENDEDOR (ID_VENDEDOR, NM_VENDEDOR) VALUES (4,
    'JOSÉ
25 BENEDITO');
26 INSERT INTO TB_VENDEDOR (ID_VENDEDOR, NM_VENDEDOR) VALUES (5,
   'DANIEL
27 FONSECA');
28
29 COMMIT WORK;
```

Listagem 6. Implementando o botão Incluir e Gravar

```
01 procedure TForm3.btnIncluirClick(Sender: TObject);
02 begin
03   DMCliente.cdsPedidos.Append;
04 end;
05
06 procedure TForm3.btnGravarClick(Sender: TObject);
07 begin
08   if DMCliente.cdsPedidos.ApplyUpdates(0) > 0 then
09     MessageDlg('Ocorreram erros na atualização,
10     Verifique!!',mtError,[mbOK],0);

11 end;
```

Configurando interface da aplicação cliente

Para concluir o nosso exemplo vamos abrir o formulário que temos em nossa aplicação cliente e construir a nossa interface. Primeiramente no formulário vamos usar a *Unit* de nosso *DataModule*, então vamos pressionar *Alt+F11* e selecionar a *Unit uDMCliente* para podermos utilizar os recursos disponíveis nela. Para facilitar a criação de nossa interface dê um duplo clique no *cdsPedidos* que se encontra no *DataModule* e através da janela de campos arraste para o formulário os seguintes campos: *ID_PEDIDO*, *DT_EMISSAO*, *VALOR_TOTAL* e *VALOR_DESC*. Ao serem arrastados serão criados automaticamente *DBEdits* na tela já ligados a esses campos. Depois faça a mesma coisa com os campos *Cliente* e *Vendedor*. Observe que para esses o *Delphi automaticamente insere controles do tipo DbLookupComboBox*.

Continuando a configuração de nossa interface vamos adicionar um *Label*, dois *BitBtn* e um *DbGrid* em nosso formulário. No *DBGrid* vamos apontar a propriedade *DataSource* dele para o *dsItemsPedidos*. Já no *Label* vamos configurar a propriedade *Caption* para *Items do Pedido* e configurar a propriedade *Caption* do primeiro *BitBtn* para *Incluir* e do segundo *BitBtn* para *Gravar*. Podemos conferir o resultado na **Figura 8.**

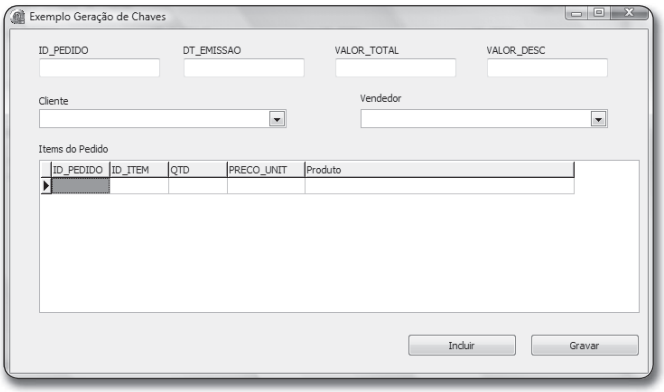


Figura 8. Design do Layout do nosso formulário de inclusão de Pedidos

Agora vamos programar o evento *OnClick* do botão *Incluir* e *Gravar*. Daremos um duplo clique nos botões e em cada evento inserimos o respectivo código, conforme **Listage**

Reparem na *Linha 8* que mesmo tendo duas tabelas só estamos chamando o método *ApplyUpdates* através da tabela *master*, pois é ela que controlará toda a gravação dos dados e envolverá o processo em uma transação. O *IF* está verificando se como retorno do *ApplyUpdates* temos algum erro gerado na gravação e caso se confirmado estamos exibindo uma mensagem informando isso ao usuário. Na *Linha 3* estamos apenas utilizando o método *Append* para incluir um novo registro na tabela. Seguindo nossa implementação vamos selecionar o nosso formulário e dar um duplo clique sobre o evento *OnCreate* do formulário e adicionar o código para abertura do *ClientDataSet* e posteriormente vamos programar o evento *OnDestroy* para realizar seu fechamento, como pode ser visto na **Listagem 7.**

Listagem 7. Implementação dos eventos OnCreate e on Destroy do formulário cliente

```
procedure TForm3.FormCreate(Sender: TObject);
begin
  DMCliente.cdsPedidos.Open;
end;

procedure TForm3.FormDestroy(Sender: TObject);
begin
  DMCliente.cdsPedidos.Close;
end;
```

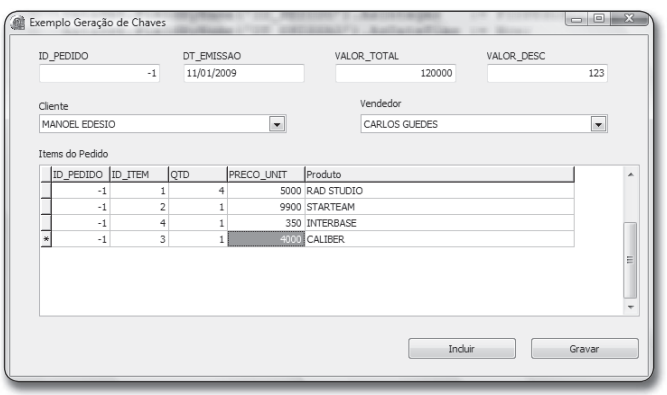


Figura 9. Incluindo Dados para o Pedido e Items de Pedido

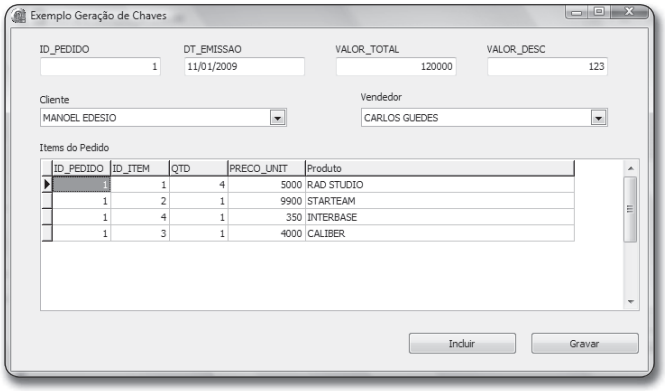


Figura 10. Visualizando o Resultado Final com os Dados já Gravados

Testando a nossa aplicação

Para verificar se a funcionalidade implementada realmente está em perfeito funcionamento vamos executar a nossa aplicação com *Ctrl+Shift+F9* clicar no botão *Incluir*. Adicione primeiramente os dados correspondentes ao *pedido* e depois os dados dos *Items do Pedido*, conforme **Figura 9.**

Vocês poderão reparar que ao incluir um novo pedido temos o código do pedido como *-1* e ao inserir os itens daquele pedido automaticamente o código *-1* é passado para todos os registros pertencentes ao detalhe para que se possa manter a integridade dos registros. Após clicarmos no botão gravar automaticamente o evento do *DataSetProvider* será disparado e teremos o nosso código executado. Com isso se após a gravação, para verificar se realmente o nosso exemplo funciona, fecharmos a aplicação e em seguida abrirmos novamente, veremos que o código foi gerado para o registro mestre e através da nossa técnica garantimos que os mesmos valores fossem passados para a tabela de detalhe, garantindo toda a integridade da geração das chaves e do relacionamento dos registros, **Figura 10**.

Conclusão

Através desse artigo certamente conseguimos perceber que os componentes *DataSetProvider* e *ClientDataSet* estão sempre nos surpreendendo. Utilizamos aqui técnicas avançadas para trabalhar com eventos no *DataSetProvider* e conseguir deixar de forma totalmente transparente para aplicação cliente a geração de chaves primárias pelo banco de dados, de forma que se consiga garantir a integridade dos dados na tabela de detalhe fazendo com que a mesma receba as informações corretas no momento da gravação do seu registro mestre. Muitas vezes nós programadores adicionamos muita complexidade a determinados processos que são mais simples do que imaginamos e com isso acabamos criando códigos que comprometem até mesmo a integridade de nossas aplicações, como é o caso da geração de chaves por plena falta de conhecimento.

Além desses recursos existem *N* funcionalidades avanças que podemos trabalhar com esses componentes. Ao aperfeiçoar o processo do lado servidor conseguimos dar um nível de profissionalismo incomparável às aplicações desenvolvidas. Espero que todos tenham gostado do artigo e nos vemos em breve no próximo, com mais recursos para ajudar vocês cada vez mais em nosso dia-a-dia. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos sobre técnicas que poderão aumentar a qualidade do desenvolvimento de software

Padrões de Projeto – Parte 2

Exemplos práticos

Nesse artigo veremos

- Quais são os padrões estruturais;
- Como implementá-los em Delphi;
- Como utilizá-los em nosso dia a dia através de exemplos práticos.

Qual a finalidade

Mostrar o uso adequado de padrões de projeto em uma aplicação Delphi.

Quais situações utilizam esse recurso?

No desenvolvimento de sistemas em geral.

Paulo Quicoli

pauloquicoli@devmedia.com.br

Editor técnico da revista ClubeDelphi, formado em processamento de dados pela FATEC-TQ. Atua como analista e desenvolvedor da ControlM Informática (www.controlm.com.br). Utiliza Delphi desde de sua primeira versão, entusiasta do desenvolvimento orientado a objetos tem publicado vários artigos sobre o assunto. Blog: www.pauloquicoli.spaces.live.com .

Na primeira parte desta série de artigos sobre *Design Patterns*, vimos como aplicar os padrões *criacionais*. Comentei que existem diversos padrões de projeto que podem ser utilizados para ajudar no desenvolvimento e todo o qualquer sistema. Comentei também que os padrões de projeto são, na verdade, conjuntos de soluções para problemas conhecidos e que em conjunto com a orientação a objetos podem fornecer recursos suficientes para a resolução de praticamente qualquer problema computacional. Neste artigo daremos continuidade ao assunto abordando os padrões estruturais que são compostos pelos seguintes:

- *Adapter;*
- *Bridge;*
- *Composite;*
- *Decorator;*
- *Façade;*
- *Flyweight;*
- *Proxy.*

Resumo do DevMan

Os padrões de projeto são soluções para problemas recorrentes que surgem no desenvolvimento de sistemas. Foram inicialmente catalogados pela GoF (Gang of Four). Quatro autores, Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides, que lançaram o livro mais famoso sobre o assunto: Design Patterns: Elements of Reusable Obejct-Oriented Software. Considerados como boa prática entre os desenvolvedores, veremos como aplicar os padrões estruturais em Delphi, em situações reais que podem acontecer no dia a dia do trabalho.

Padrão Adapter

Em sua definição o padrão *Adapter* possui a tarefa de converter a interface de uma classe para outra interface que uma classe cliente espera, permitindo que classes trabalhem juntas, mesmo possuindo interfaces diferentes. O próprio *Delphi* aplica automaticamente o padrão *Adapter* para nós. Em outras palavras algumas vezes, uma classe projetada para ser reusada não o é porque sua interface não combina com a interface requisitada por uma aplicação de um domínio específico, um exemplo disso: uma classe *TextView* existente poderia ser reusada através de uma interface *TextShape* em um editor de desenhos para desenhar e organizar elementos gráficos (linhas, polígonos, etc.), tratando o texto como mais um tipo de elemento gráfico (**Figura 1**).

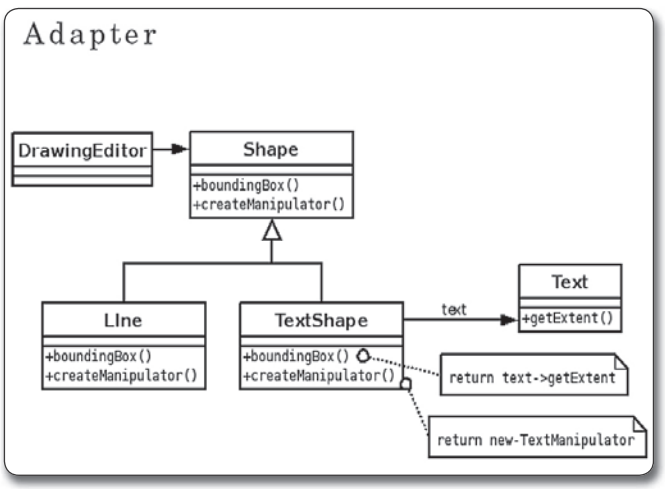
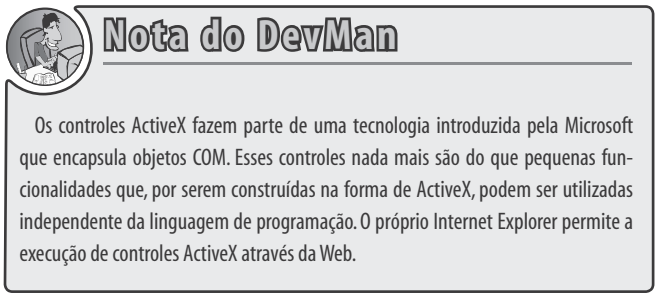


Figura 1. Exemplo do padrão Adapter em uso

Implementação

Para que possamos entender perfeitamente o que é e como utilizar o padrão Adapter, vamos desenvolver uma aplicação capaz de ler um arquivo PDF. Por isso inicie um novo projeto *Win32* no *RAD Studio Delphi 2007* e salve-o como *Adapter*. Observe que um *Project Group* é criado por padrão. Vamos utilizá-lo para manter posteriormente os outros exemplos que virão. Desta forma vocês terão um catálogo para consulta. Vamos salvar o *Project Group* como *Estruturais*. Para isso clique com o botão direito do mouse sobre ele e escolha *Save Project Group*.

Vamos clicar agora no menu *Component>Import Component*. Na janela que surgir vamos escolher *Import ActiveX Control*. Esses controles possuem tipos diferentes do *Delphi* e ao importá-los uma classe *Adapter* também é criada e ela é responsável em fazer com que o controle se comunique com nossa aplicação. Ao clicar em *Next* podemos escolher qual controle importar. Neste exemplo vamos criar uma pequena aplicação que permita a leitura de arquivos *PDF* como já mencionado. Portanto escolhemos com um clique duplo o *Adobe Acrobat 7.0 Browser Control Type*, como pode ser visto na **Figura 2**.



Nota do DevMan

Os controles ActiveX fazem parte de uma tecnologia introduzida pela Microsoft que encapsula objetos COM. Esses controles nada mais são do que pequenas funcionalidades que, por serem construídas na forma de ActiveX, podem ser utilizadas independente da linguagem de programação. O próprio Internet Explorer permite a execução de controles ActiveX através da Web.

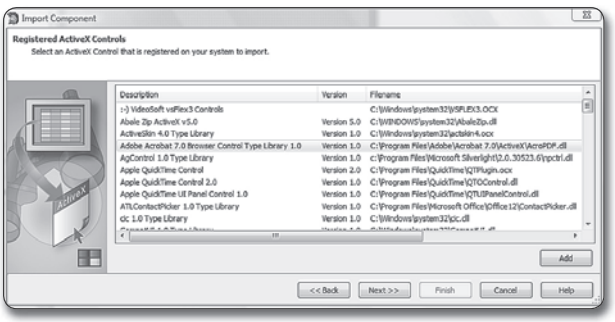


Figura 2. Lista de controles ActiveX disponíveis

Nota: O controle *Active Acrobat 7.0 Browser Control Type Library 1.0* estará disponível apenas se em sua máquina o *Adobe Acrobat Reader* estiver instalado.

Veja na **Figura 3** que o *Delphi* cria para nós uma classe *Adapter* em forma de componente, também conhecida como *Wrapper*, que irá permitir o uso do *ActiveX* utilizando a linguagem *Delphi*.

Ao clicarmos em *Next* e depois em *Finish* a classe gerada será aberta e poderemos ver sua implementação e então podemos instalar o componente criado. Para instalar o componente é preciso adicionar a *Unit* a um *package*. Vamos então criar o pacote, acessando o menu *File>New>Package Delphi for Win32*. No pacote criado clicamos com o botão direito do mouse e escolhemos *Add* para localizar a *Unit* anteriormente criada. Após isso podemos novamente clicar com o botão direito do mouse sobre o pacote

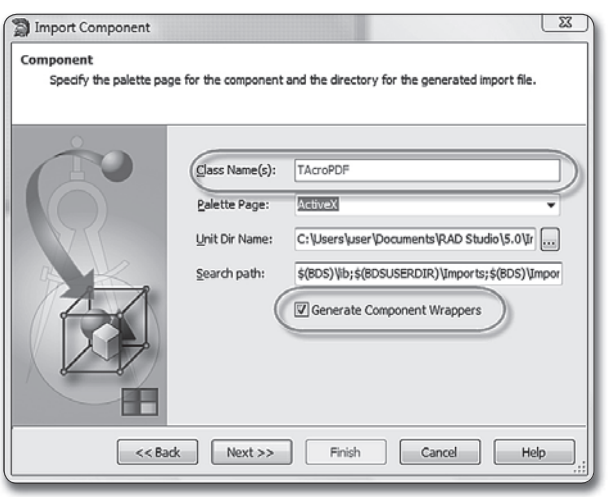


Figura 3. Criação da classe Adapter

Nota do DevMan

Em 1993 a Microsoft quis padronizar a forma com que os softwares pudessem ser organizados, de forma que componentes pudessem ser criados e reutilizados entre as mais diversas aplicações. Surge assim uma interface que permite a criação dinâmica de objetos e comunicação entre aplicações, o componente Object Model, ou simplesmente COM. Essa tecnologia serviu de base para diversas outras como o ActiveX, OLE, DCOM e COM+. Mesmo que tenha sido desenvolvida na forma de interface, permitindo sua implementação nos mais variados sistemas operacionais, o COM é principalmente utilizando no mundo Windows.

e escolher *Install* e o componente *TAcroPDF* irá aparecer na guia *ActiveX*. Arraste para o formulário principal um controle *TAcroPDF*, um *TPanel* e um *OpenDialog*. Insira dentro do *TPanel* um *TButton* deixando-os organizados como na **Figura 4**.

Figura 4. Formulário principal

No evento *onClick* do botão adicione o seguinte código:

```
if OpenDialog1.Execute then
begin
  AcroPDF1.LoadFile(OpenDialog1.FileName);
end;
```

Veja que temos o método *LoadFile* implementado no componente *AcroPDF*. Esse método por sua vez chama a rotina do controle *ActiveX* responsável por carregar um arquivo *.PDF*. Execute a aplicação e carregue um arquivo para ver tudo isso funcionando.

Como podemos ver, a utilização do padrão Adapter é muito simples e não exige nenhum grande estudo para colocar logo em prática.

Listagem 1. Definição das classes de Log

```
01  TLog = class
02    procedure Gravar(Texto: string); virtual; abstract;
03  end;
04
05  TLogFormatoTXT1 = class(TLog)
06  private
07    FStringList: TStringList;
08  public
09    constructor Create; reintroduce;
10    destructor Destroy; override;
11    procedure Gravar(Texto: string); override;
12  end;
13
14  TLogFormatoTXT2 = class(TLog)
15  private
16    FStringList: TStringList;
17  public
18    constructor Create; reintroduce;
19    destructor Destroy; override;
20    procedure Gravar(Texto: string); override;
21  end;
```

Padrão Decorator

O *Decorator* pode adicionar responsabilidades adicionais a um determinado objeto. É uma alternativa à criação de heranças na implementação de novos recursos. Vamos criar um exemplo onde temos um classe que pode registrar erros encontrados no sistema só que também deve permitir ao usuário enviar esse erro por email.

Adicione ao *Project Group* um novo projeto *VCL Forms* e salve-o como *Decorator*. No formulário principal desse projeto vamos adicionar as classes de log, conforme **Listagem 1**.

Veja que definimos uma classe base, *TLog* e que herdamos dela duas classes, *TLogFormatoTXT1* e *TLogFormatTXT2*, que simplesmente gravam uma mensagem passada como parâmetro para um arquivo texto. Isso pode ser visto na **Listagem 2**.

Listagem 2. Gravando o log em formatos diferentes

```
01 procedure TLogFormatoTXT1.Gravar(Texto: string);
02 begin
03   FStringList.Append(Texto);
04   FStringList.SaveToFile('log.txt');
05 end;
06
07 procedure TLogFormatoTXT2.Gravar(Texto: string);
08 begin
09   FStringList.Append(DateToStr(Now)+ ' Erro: '+ Texto);
10   FStringList.SaveToFile('logFormato2.txt');
11 end;
```

Nota: Os detalhes completos da implementação de ambas as classes não serão mostrados aqui por que foge do escopo do artigo, contudo, o código fonte completo está disponível para download.

Até o momento então criamos nossa rotina de log que permite salvar em formatos diferentes uma mensagem de erro. Para utilizá-las, adicione um *TButton* no formulário principal e no seu evento *onClick* adicione o código da **Listagem 3**.

Imagine que agora ela tenha que enviar também por email essa mensagem. Poderíamos implementar essa rotina na classe base *TLog*, o que implicaria em mudar a estrutura de nossa herança, visto que um possível novo método *EnviarPorEmail* deveria ser criado. Poderíamos também criar uma nova herança a partir de *TLog*, mas imagine se um novo recurso deve ser implementado, vamos criar outra herança? Um dos fundamentos da orientação a objetos é reutilização de código e cabe a nós utilizar isso de tal forma que não implique em uma cascata de alterações. Por exemplo, nossa classe *TLog* pode ser utilizada por um sistema inteiro, então temos que pensar em uma forma de adicionar um novo recurso sem ter que alterar

Listagem 3. Utilizando o Log

```
procedure TDecoratorF.Button1Click(Sender: TObject);
var
  Log: TLog;
begin
  Log := TLogFormatoTXT1.Create;
  Log.Gravar('Texto da mensagem');
  FreeAndNil(Log);
end;
```

em muito todas suas chamadas. Então o padrão *Decorator* vem ao nosso socorro. Logo após a declaração da classe *TLogFormatoTXT2* adicione o código da **Listagem 4**.

Listagem 4. Estrutura do Decorator

```
01  TDecoratorLog = class(TLog)
02  private
03    FLog: TLog;
04  public
05    constructor Create(Log:TLog); reintroduce;
06    destructor Destroy; override;
07    procedure Gravar(Texto: string); override;
08  end;
```

Com a classe *TDecoratorLog* criamos uma estrutura básica que servirá para *decorar* qualquer instância de *TLog* com a funcionalidade que desejarmos, ou seja, toda nova funcionalidade que for preciso adicionar à *TLog* deve herdar de *TDecoratorLog*. Na **Listagem 5** vemos sua implementação.

Listagem 5. Implementação de TDecoratorLog

```
01 constructor TDecoratorLog.Create(Log: TLog);
02 begin
03   FLog := Log;
04 end;
05
06 destructor TDecoratorLog.Destroy;
07 begin
08   FLog.Free;
09   inherited;
10 end;
11
12 procedure TDecoratorLog.Gravar(Texto: string);
13 begin
14   FLog.Gravar(Texto);
15 end;
```

Veja que simplesmente passamos a execução dos métodos à instância de *TLog* armazenada no campo *FLog*. Agora vamos criar o recurso de enviar por email, ou seja, vamos criar o *decorator* real. Adicione logo abaixo da declaração da classe *TDecoratorLog* o código da **Listagem 6**.

Observe que é em *TLogPorEmail* que adicionamos os detalhes do envio por email. Veja que criamos um campo que armazena o email padrão. Na **Listagem 7** vemos como isso é utilizado.

No construtor da classe definimos o email padrão, e no método *Gravar*, simplesmente chamamos o *Gravar* original e em seguida adicionamos o código que enviaria o texto por email. Para colocar tudo isso em prática seria necessário mudar apenas a chamada de criação da rotina de log, como pode ser visto na **Listagem 8**.

Execute e veja. Quanto ao uso do *Decorator* podemos concluir que facilmente podemos adicionar novos recursos a uma determinada classe sem alterar em demasia todas suas chamadas. Se fosse necessário criar, por exemplo, um novo recurso que imprimisse esse log assim que fosse registrada uma mensagem, bastaria criar uma nova classe que herde de *TDecoratorLog* e nela criar o código de impressão.

Listagem 6. Decorator de envio por email

```
01  TLogPorEmail = class(TDecoratorLog)
02  private
03    FEmailPadrao: string;
04  public
05    constructor Create(Log: TLog); reintroduce;
06    procedure Gravar(Texto: string); override;
07  end;
```

Listagem 7. Implementando o decorator

```
01 constructor TLogPorEmail.Create(Log: TLog);
02 begin
03   inherited Create(Log);
04   FEmailPadrao := 'pauloquicoli@devmedia.com.br';
05 end;
06
07 procedure TLogPorEmail.Gravar(Texto: string);
08 begin
09   inherited Gravar(Texto);
10   ShowMessage('Enviando log para ' + FEmailPadrao);
11 end;
```

Listagem 8. Utilizando o decorator

```
01 procedure TDecoratorF.Button1Click(Sender: TObject);
02 var
03   Log: TLog;
04 begin
05   Log := TLogPorEmail.Create(TLogFormatoTXT1.Create);
06   Log.Gravar('Texto da mensagem');
07   FreeAndNil(Log);
08 end;
```

Padrão Façade

O *Façade* é um padrão muito comum, que é utilizado sem nos darmos conta. Ele possui a responsabilidade de simplificar um subsistema, tornando fácil o acesso às suas rotinas. Ele resume uma grande e complexa rotina em simples métodos. Também chamado de *Serviço*, podemos ter várias implementações dele em um sistema, sendo o ideal organizá-las por funcionalidade. Por exemplo, um serviço de autenticação, serviço de impressão e outros que forem necessários.

Nosso exemplo de *Façade* será bem simples. Teremos uma tela de pesquisa que irá localizar um determinado nome contido em um *TClientDataSet*. O *Façade* em si será uma classe separada que realiza a operação de busca, aplicando antes toda validação necessária.

Adicione ao *ProjectGroup* um novo projeto *VCL Forms*. Salve-o como *Facade*. Em seu formulário principal adicione um *TClientDataSet*, *TDataSource*, *TDBDGrid*, *TLabel* e um *TEdit*. Crie no *ClientDataSet* um campo chamado *Nome* do tipo *string*, tamanho 50. Ligue o *DataSource* a ele e o *DBGrid* ao *DataSource*. Organize o formulário para que fique como na **Figura 5**.

Agora vamos criar duas exceções personalizadas que serão posteriormente utilizadas pelo nosso *façade*. Criamos então antes da declaração do formulário como no código a seguir e também declaramos nossa classe de serviço.

```
TENomeVazio = class(Exception);
TENaoEncontrado = class(Exception);
TConsultaFacade = class;
```

Como já foi mencionado, o formulário fará uso da classe de busca. Portanto adicionamos na seção *private* do formulário um campo do tipo *TConsultaFacade* e no evento *onCreate* do formulário o instanciamos e inserimos alguns registros no *ClientDataSet*, como pode ser visto na **Listagem 9**.

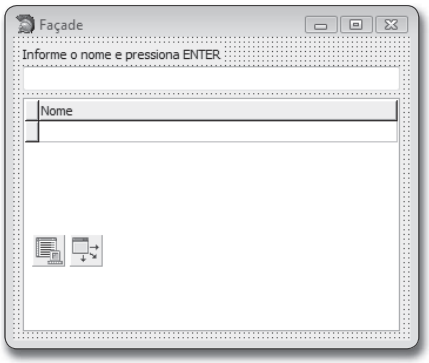


Figura 5. Disposição dos controles

Chegou a vez de implementarmos a classe *TConsultaFacade*. Após a declaração da classe do formulário inserimos o código a seguir:

```
TConsultaFacade = class
public
  procedure Consultar(Nome: string);
end;
```

Também é preciso implementar o método *Consultar*. Ele é quem simplifica para o formulário a complexidade de se realizar uma busca. Veja a **Listagem 10**.

Veja que caso o nome a ser consultado não for informado, disparamos a exceção adequada, que previamente

Listagem 9. Create do formulário

```
01 procedure TFacadeF.FormCreate(Sender: TObject);
02 begin
03   ServicoConsulta := TConsultaFacade.Create;

04   cdsNomes.CreateDataSet;
05
06   cdsNomes.Insert;
07   cdsNomesNome.AsString := 'Paulo Quicoli';
08   cdsNomes.Post;
09
10   cdsNomes.Insert;
11   cdsNomesNome.AsString := 'José da Silva';
12   cdsNomes.Post;
13
14   cdsNomes.Insert;
15   cdsNomesNome.AsString := 'Carla Quicoli';
16   cdsNomes.Post;
17
18   cdsNomes.Insert;
19   cdsNomesNome.AsString := 'Mariano Cunha';
20   cdsNomes.Post;
21
22   cdsNomes.Insert;
23   cdsNomesNome.AsString := 'Eliberto de Abreu';
24   cdsNomes.Post;
25
26   cdsNomes.Insert;
27   cdsNomesNome.AsString := 'Ricardo Carlos de Albuquerque';
28   cdsNomes.Post;
29 end;
```

Listagem 10. Implementando a consulta

```
01 procedure TConsultaFacade.Consultar(Nome: string);
02 begin
03   FacadeF.cdsNomes.First;
04   if Trim(nome) = '' then
05     raise TENomeVazio.Create('O nome a se buscar não pode ser
         vazio');
06
07   if not
08      FacadeF.cdsNomes.Locate('Nome',Nome,[loPartialKey,
         loCaseInsensitive]) then
09        raise TENaoEncontrado.Create('Não foram encontrados
            resultados para ' +
10              Nome);
11 end;
```

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora mesmo o portal do assinante ClubeDelphi e assista uma vídeo aula de Everson Borges Volaco que mostra como criar e tratar exceções dentro de um banco de dados Firebird

http://www.devmedia.com.br/articles/viewcomp.asp?comp=4842

declaramos. O mesmo acontece caso não seja encontrado o nome informado. Posteriormente tratamos isso no formulário, como pode ser visto no evento *onKeyUp* do *Edit1*, na **Listagem 11**.

Se compararmos a **Listagem 10** e a **Listagem 11** podemos ver que o *façade* ocultou do formulário algumas exigências para que uma consulta possa ser realizada, tornando assim seu acesso mais direto. Por exemplo, o formulário não sabe que antes de qualquer busca é preciso enviar a posição atual do *ClientDataSet* para o primeiro registro. Essa é a função de um *Façade* ou *Service*, ocultar uma complexidade, tornando seu acesso mais direto e simples o possível.

Listagem 11. Utilizando o Façade

```
01 procedure TFacadeF.Edit1KeyUp(Sender: TObject; var Key: Word;
02   Shift: TShiftState);
03 begin
04   if Key = VK_RETURN then
05   begin
06     try
07       ServicoConsulta.Consultar(Edit1.Text);
08     except
09       on e: TENomeVazio do
10       begin
11         ShowMessage(e.Message);
12         Edit1.SetFocus;
13       end;
14       on e:TENaoEncontrado do
15       begin
16         Application.MessageBox(Pchar(E.Message),'Atenção',MB_OK +
17                MB_ICONWARNING);
18
19       end;
20     end;
21   end;
22 end;
```

Conclusão

Nesta segunda parte do artigo vimos que os padrões podem nos ajudar a preparar nosso sistema para futuras mudanças de requisitos, deixando o código mais *maleável*. Além disso, seu uso pode até mesmo passar despercebido como é o caso do padrão *Façade*. Como já foi dito antes, os padrões são um conjunto de soluções já testadas, não são nada *novo*, apenas foram catalogados. Por isso não estranhe ao se deparar com um código de um determinado padrão e você dizer: *"Olha, eu já usava isso faz tempo..."*.

Para o próximo artigo veremos os padrões comportamentais, fique ligado.

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos sobre técnicas que poderão aumentar a qualidade do desenvolvimento de software

Testes Automatizados

Exemplos práticos utilizando DUnit

Nesse artigo veremos

- Como automatizar alguns testes do sistema;
- Como implementá-los utilizando o DUnit;
- Como utilizá-los em nosso dia a dia através de exemplos práticos.

Qual a finalidade

Garantir a qualidade de uma aplicação Delphi.

Quais situações utilizam esse recurso?

Todo e qualquer sistema pode e deve passar pela fase de testes unitários para garantir sua integridade.

Resumo do DevMan

A qualidade de um software deve ser garantida durante seu ciclo de vida, mesmo em períodos de constantes alterações. Como podemos assegurar que novas implementações não venham quebrar algum código já existente? Aplicando testes unitários. Os testes unitários pode verificar as rotinas de um sistema de forma automática, desde que o sistema ofereça uma arquitetura que permita sua aplicação.

Neste artigo veremos como é essa arquitetura e como criar e utilizar os testes utilizando o framework DUnit, em uma aplicação simples, mas que é bem próxima do dia-a-dia do desenvolvedor Delphi.

Paulo Quicoli

pauloquicoli@devmedia.com.br

Editor técnico da revista ClubeDelphi, formado em processamento de dados pela FATEC-TQ. Atua como analista e desenvolvedor da ControlM Informática (www.controlm.com.br). Utiliza Delphi desde de sua primeira versão, entusiasta do desenvolvimento orientado a objetos tem publicado vários artigos sobre o assunto. Blog: www.pauloquicoli.spaces.live.com .

Durante o desenvolvimento de um software vários requisitos podem ser alterados exigindo que o sistema também se adéqüe. O quão difícil, ou fácil, será implementar os ajustes em um sistema já em funcionamento que depende da estrutura/arquitetura utilizada. A utilização de testes automáticos pode funcionar como uma espécie de *seguro* que pode garantir a estabilidade do seu código durante uma fase de alterações. Tudo isso para evitar a situação do famoso ditado: *em time que está ganhando, não se mexe*.

Esse artigo trata de como aplicar testes unitários em uma aplicação comum que possua o mínimo de arquitetura.

Pensando em arquitetura

Todo sistema informatizado encapsula uma série de lógicas de programação. Essas lógicas por sua vez podem ser agrupadas, divididas em partes distintas umas das outras. Temos regras para exibição de dados, regras que determinam como esses dados devem ser salvos e por fim, temos a regra do negócio propriamente dita. Sendo assim, podemos dizer que um sistema pode possuir *camadas* e observando seu conteúdo chegamos à **Figura 1**.

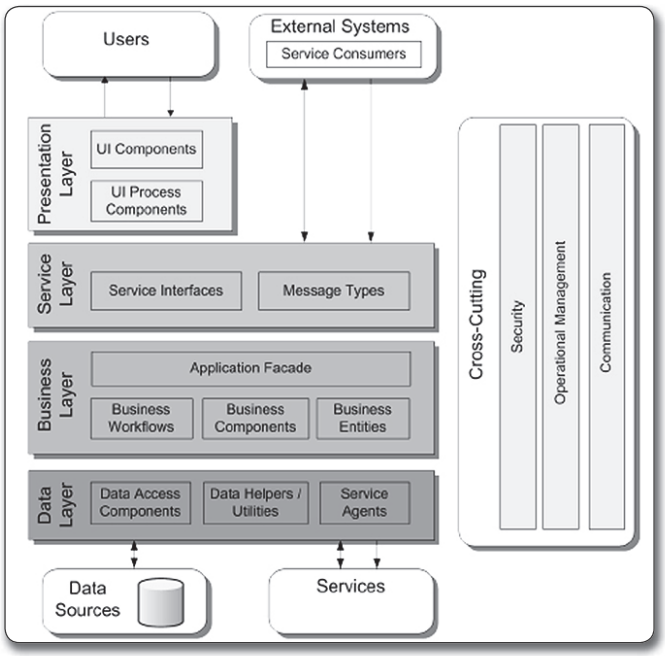


Figura 1. Camadas de software

• *Camada de apresentação:* É através dela que os usuários interagem com o sistema. Não importa se é *Desktop* ou *WEB*. É responsável em saber como apresentar os dados, como permitir sua entrada e como exibir adequadamente as respostas do sistema. Interage com a camada de serviço;
• *Camada de serviço:* Disponibiliza ações que representam as funcionalidades do sistema, como *Busca de clientes por Nome* ou *Impressão do relatório de vendas*. Esta por sua vez se comunica com a camada de negócios;
• *Camada de negócios:* Aqui é onde suas entidades estão organizadas. Temos classes que representam o seu negócio, como *Cliente, Produto, Venda*, etc. Todos os dados necessários são obtidos da camada de dados;
• *Camada de dados:* Esta camada é responsável por fazer o acesso ao banco de dados, recuperando e atualizando informações. Tudo que diz respeito a banco de dados deve ser centralizado nessa camada.

A comunicação necessária entre as camadas ocorre sempre de cima para baixo. Por exemplo, a camada de dados não precisa, e nem é recomendável, saber algo sobre a camada de apresentação. Pensando em um exemplo mais aplicado à realidade, por que que em seu *DataModule* existiria uma referência ao formulário principal do sistema? Aplicar corretamente essa separação não é um trabalho difícil, visto que mesmo separadas as camadas são *ligadas* entre si pelos *Design Patterns*.

Mas se as camadas devem ficar separadas, como que podem ao mesmo tempo estar ligadas? Para entender melhor a idéia, vamos a um exemplo bem simples. Imagine um caderno *aspiral*, daqueles que você leva ou levava para a faculdade. Cada folha é única, separada uma da outra, porém, interligadas através da *aspiral*. As camadas são as folhas, que na prática são utilizadas sempre para escrever do início para o fim, ou seja, sempre da primeira para a próxima. O arame *aspiral* representa os vários *design patterns* existentes para formar o *caderno*, que é a aplicação final.

Se olharmos novamente para a **Figura 1** vemos que existe uma outra categoria de camada, chamada *cross-cutting*, que traduzida de forma livre seria *atravessadora*. As classes que fazem parte dessa camada podem interagir com todas as outras porque representam necessidades de *infraestrutura*. Um exemplo disso é uma implementação de *log* que registra as ações do usuário na camada de interface, depois registra os dados que estão sendo manipulados na camada de dados.

Testes unitários

Testes unitários são aqueles que verificam de forma automatizada se uma determinada funcionalidade de uma classe está funcionando como deveria. Esses testes devem agir sobre funcionalidades únicas, por isso o nome *unitário*. Sua origem está associada à técnica de desenvolvimento *TDD* (*Test Driven Development)* que significa *desenvolvimento digirido a testes*. Nessa técnica, ou, até por que não dizer *ideologia*, todo um sistema deve ter suas classes, funcionalidades submetidas a testes unitários. Mas para o *TDD* os testes começam antes mesmo de se ter um código a testar. Parece estranho, mas como diz o próprio nome os testes dirigem o desenvolvimento. Quero manter o foco na aplicação dos testes, portando vamos primeiramente ter código para depois testá-lo.

Infelizmente no dia a dia dos desenvolvedores *Delphi* esse tipo de teste não é muito utilizado e por muitos é desconhecido. Porém ao observar outras comunidades como *Java* ou *.NET* essa já é uma prática comum. Essas comunidades enxergaram os benefícios de sua utilização. Posso citar três benefícios, *aumento de segurança*, *confiabilidade* e *distribuição do aprendizado*.

Resumo do DevMan

Os padrões de projeto são soluções para problemas recorrentes que surgem no desenvolvimento de sistemas. Foram inicialmente catalogados pela GoF (Gang of Four). Quatro autores, Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides, que lançaram o livro mais famoso sobre o assunto: Design Patterns: Elements of Reusable Obejct-Oriented Software.

• *Aumento de segurança:* Esta segurança diz respeito ao desenvolvedor que se sentirá amparado para realizar qualquer alteração no código fonte da aplicação. Isso porque ao terminar sua alterações, um teste geral do sistema pode ser executado de forma automática e qualquer falha originada pela nova alteração *pode* ser descoberta. Resumindo, o teste irá mostrar onde sua nova alteração está causando problema. Com isso o desenvolvedor ganha em agilidade para corrigir a situação;
• *Confiabilidade:* Com garantia de que está tudo funcionando, podemos dizer que seu código é confiável podendo até chegar à situação de ser livre de *bugs*;
• *Aprendizado:* Os testes, por simular as funções do sistema, podem ser utilizados como uma forma de estudo para novos membros da equipe. Isso porque em um teste unitário você precisa cercar todas as situações possíveis de falha. Com isso, a forma que uma determinada rotina funciona fica ali demonstrada.

Antes de vermos como aplicá-los vamos conhecer a aplicação a ser testada.

Aplicação para demonstração

Essa aplicação é bem simples. Constitui-se de um cadastro de clientes (*id, nome, email*) e sua consulta. Mantemos essas informações em um *ClientDataSet* temporário, ou seja, não faremos acesso a banco algum, tudo permanece em memória. Isso, para simplificar o sistema. Porém nele foi aplicada uma arquitetura que *permite* a criação dos testes. Com isso temos as divisões vistas na **Figura 2**.

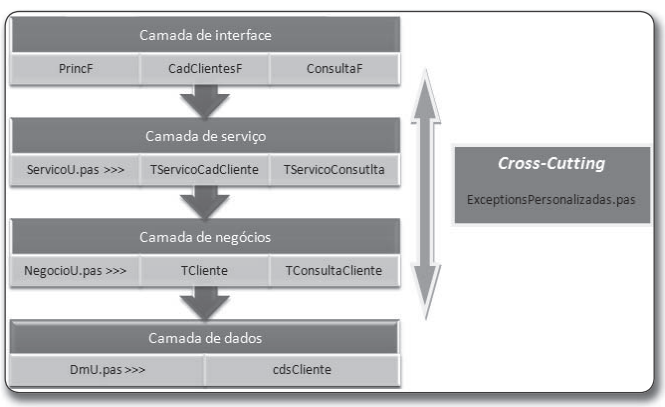


Figura 2. Arquitetura do programa demo

De uma forma simples então temos na camada de apresentação três formulários onde, *CadClientesF* e *ConsultaF* interagem com a camada de serviço que é representada na *Unit Servico.pas* que contém duas classes, *TServicoCliente* e *TServicoConsulta*. Esta camada disponibiliza para a interface as *ações* a serem realizadas, que por sua vez delega à camada de negócio centralizada na *Unit NegocioU.pas*, a execução dessas *ações*. E por fim as classes *TCliente* e *TConsultaCliente*, da camada de negócios, utiliza os métodos disponibilizados pela camada de dados, que aqui é implementada no *DataModule DmU.pas* e pelo seu controle *TClientDataSet cdsCliente*.

Nota: Por se tratar de um exemplo simples, a construção do exemplo não será explorada nesse artigo. Seu código está disponível para download no site da DevMedia.

Criando os testes

Como já foi mencionado, os testes são executados de forma automática. Logo isso implica que algum outro programa irá executá-los. Esse programa na verdade é um *framework* que oferece meios de criação e execução de testes unitários. Existem diferentes versões para os diversos ambientes de desenvolvimento. Para o *Delphi* temos o *framework DUnit*. Para iniciar é preciso realizar seu *download* no *site https://sourceforge.net/projects/dunit/.*

Terminado o *download* apenas descompacte o arquivo para alguma pasta de fácil acesso. Não precisa instalar nada. Vamos agora abrir no *Delphi* o projeto do programa de teste e definir em suas propriedades a localização do *DUnit*. Para isso, vamos em *Project>Options>Directories/Conditionals* e no campo *Search Path* informamos a localização, como pode ser visto na **Figura 3**.

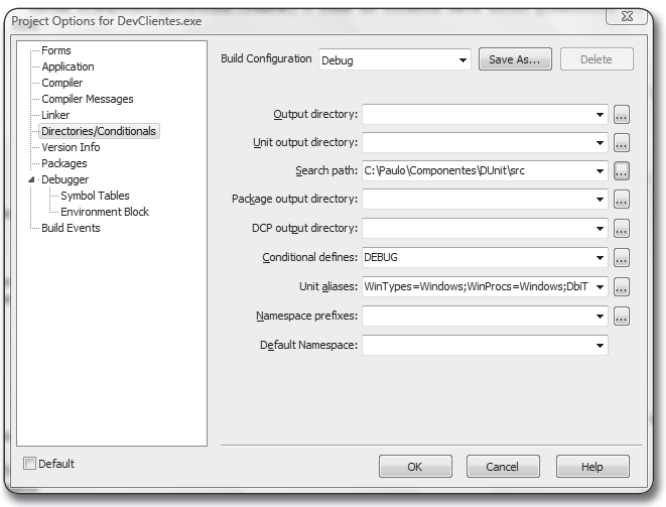


Figura 3. Informando a localização do DUnit

Criamos então uma nova aplicação, removemos o formulário criado automaticamente pelo *Delphi* e a salvamos na mesma pasta do projeto a ser testado, ou seja, na mesma pasta do projeto de demonstração. É interessante nomeá-lo de forma que indique que ele contém os testes. Um formato muito utilizado é o nome do projeto a ser testamos mais o sufixo *Tests*. Por exemplo, se o programa a testar chama-se *DevClientes*, o projeto de testes será *DevClientesTests*.

Nota: Pode-se utilizar as funcionalidades da janela *Project Manager* para agrupar os projetos em um único *Project Group*.

Agora vamos abrir o fonte do projeto de teste, acessando *Project>View Source* e adicionar as referências ao *DUnit*, como é mostrado na **Listagem 1**.

Listagem 1. Configurando o projeto para o uso do DUnit

```
01 program DevClientesTests;
02
03 uses
04   Forms,
05   TestFrameWork,
06   GUITestRunner;
07
08 {$R *.res}
09
10 begin
11   Application.Initialize;
12   Application.MainFormOnTaskbar := True;
13   GUITestRunner.RunRegisteredTests;
14 end.
```

As *Units TestFrameWork* e *GUITestRunner* informadas na seção *Uses* fazem parte do *DUnit*, sendo que a primeira disponibiliza as rotinas para construção de testes e a segunda permite a execução e apuração dos resultados de forma visual em um formulário que é criado em tempo de execução. Com isso configurado estamos prontos para iniciar, mas antes é preciso entender a estrutura de teste.

Conhecendo a estrutura básica de teste

Os testes são desenvolvidos por nós mesmos. Eles nada mais são do que classes que possuem métodos representando cada teste ser efetuado, ou seja, vamos escrevê-los propriamente dito. Podemos organizá-los por módulos que são chamados de *TestCase*. Vamos criar então nosso primeiro *TestCase* para verificar se a consulta está funcionando.

Adicione uma nova *Unit* ao projeto e na seção *Uses* adicione *TestFramework*. Salve como *DevClientesTestCases*.Agora adicionamos uma nova classe, como no código a seguir:

```
type
 TTestCaseConsulta = class(TTestCase)
 published
   procedure TestNomeEncontrado;
 end;
```

Observe que a classe *TTestCaseConsulta* está representando tudo o que será aplicado no módulo de consulta. Cada método representa um teste a ser realizado. Vamos aplicá-los a camada de negócios. Então é preciso adicionar na seção *Uses* referências a *dmU* e *NegocioU*. Implemente o método *TestNomeEncontrado* como na **Listagem 2**.

Nota: Também é preciso adicionar referência a *SysUtils* e *Forms*. Use o *Alt + F11* para adicionar as *Units* necessárias.

Listagem 2. Testando a localização

```
01 procedure TTestCaseConsulta.TestNomeEncontrado;
02 var
03   consulta: TConsultaCliente;
04   NomeABuscar: string;
05 begin
06   NomeABuscar := 'Barak Obama Hussein';
07   Application.CreateForm(Tdm, dm);
08
09   consulta := TConsultaCliente.Create;
10   consulta.Consultar(NomeABuscar);
11   CheckTrue(dm.cdsClientesNome.AsString = NomeABuscar);
12
13   FreeAndNil(consulta);
14   FreeAndNil(dm);
15 end;
```

Veja que é preciso instanciar o *DataModule*, temos que fazer tudo como se fosse uma execução normal do sistema. Como vamos realizar testes em rotinas que manipulam uma fonte de dados, o ideal é preparar essa fonte com dados previsíveis, para que possamos testar. Em nosso caso eu já sabia que existe no *cdsClientes* um registro que contém o nome que foi buscado, isto porque no evento *OnCreate* do *DataModule* são criados três registros padrão.

O ponto a se destacar aqui é a chamada do método *CheckTrue*. Este é um dos métodos oferecidos pelo *framework DUnit*. Ele verifica se uma condição é verdadeira. Se for, a execução passa no teste, caso contrário o teste retorna falso. Existem outros métodos disponíveis, ao longo do artigo veremos outros.

Antes de executarmos o teste é preciso registrá-lo no *DUnit* para que ele saiba que existe um *TestCase* a ser avaliado. Para isso criamos a seção *Initialization* e nela inserimos o seguinte código:

```
initialization
  TestFramework.RegisterTest(TTestCaseConsulta.Suite);
```

Agora sim, compile o projeto *DevClintestTests* e o execute, a interface visual do *DUnit* irá aparecer, como na **Figura 4**.

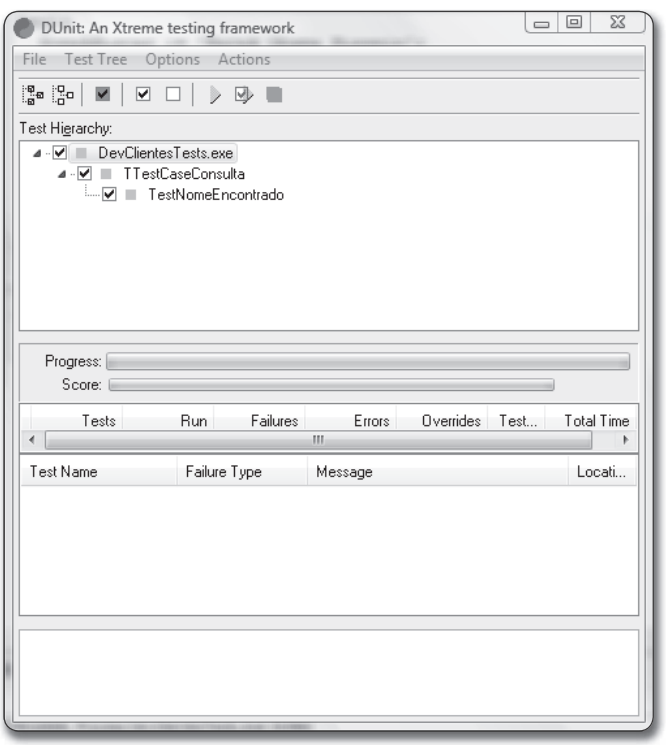


Figura 4. Interface para execução de testes

Ao clicar no botão *play* ou pressionar *ENTER*, os testes são executados e o resultado pode ser visualizado como na **Figura 5**.

Cada teste bem sucedido é sinalizado com a cor verde, os não sucedidos em vermelho. Observe também na **Figura 5** que é mantido um placar, *score*, com a porcentagem total

do testes que passaram. Vamos continuar porque existem mais situações a se testar na consulta.

Vamos verificar se a consulta está se comportando como deve ao não encontrar um registro. Caso um registro não seja encontrado pela consulta, o *cdsCliente* deve permanecer posicionado no primeiro registro, que neste caso possui o nome *Paulo Quicoli*.

Adicionemos então um novo método *TestNomeNaoEncontrado* na classe *TTestCaseConsulta* e o implementemos como na **Listagem 3**.

Listagem 3. Método TestNomeNaoEncontrado

```
01 procedure TTestCaseConsulta.TestNomeNaoEncontrado;
02 var
03   consulta: TConsultaCliente;
04   NomeABuscar: string;
05 begin
06   NomeABuscar := 'Bin Laden';
07   Application.CreateForm(Tdm, dm);
08
09   consulta := TConsultaCliente.Create;
10   consulta.Consultar(NomeABuscar);
11
12   CheckFalse(dm.cdsClientesNome.AsString = NomeABuscar);
13   CheckTrue(dm.cdsClientesNome.AsString = 'Paulo
       Quicoli','Não ficou
14             posicionado no primeiro registro');
15
16   FreeAndNil(consulta);
17   FreeAndNil(dm);
18 end;
```

O código do teste é bem similar ao da **Listagem 2,** porém verifique as *linhas 12* e *13*. Na *linha 12* verificamos se o resultado é falso. Isso comprova que não foi encontrado algum registro com o nome informado. Na *linha 13* verificamos se o registro posicionado é o primeiro, comparando os nomes. Observe que adicionamos um parâmetro a mais na chamada do método *CheckTrue*. Esse texto será exibido caso o teste falhe. Repare que estamos sempre criando o *DataModule* no início do teste e ao terminá-lo o estamos destruindo. Esse é código repetitivo, porém necessário. Para agilizar a criação dos testes o *DUnit* disponibiliza dois métodos especiais, *Setup* e *TearDown*.

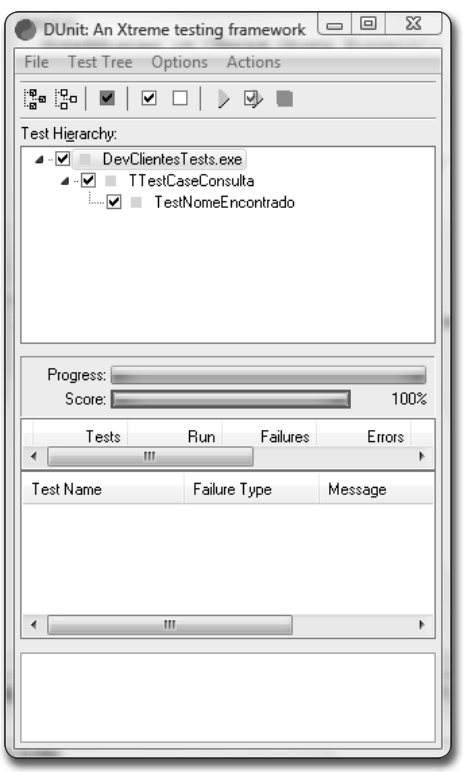


Figura 5. Testes realizados

O primeiro é executado antes da execução do teste e o segundo em sua finalização. Com isso podemos colocar o código de criação do *DataModule* no método *Setup* e para destruí-lo codificamos no *TearDown*. Também podemos fazer isso com a criação do objeto de consulta, mas no caso dele é preciso criar um campo privado para armazenar o objeto. Vamos então realizar um *override* (*sobreposição*) desses métodos e dar uma enxugada no código. Na **Listagem 4** temos o resultado final.

Nas *linhas 17 à 29* centralizamos nos métodos *Setup* e *TearDown* tudo o que um teste de consulta precisa para ser executado e que era repetitivo. Veja no método *TestNomeNaoEncontrado* que não fazemos mais referência à criação do *DataModule* e do objeto *consulta*. Após esta organização no código ainda falta um último teste para a consulta.

Na camada de negócio é verificada uma quantidade de caracteres que é passada no nome a ser localizado, o mínino são três. Caso

Listagem 4. Utilização dos métodos Setup e TearDown

```
01 type
02  TTestCaseConsulta = class(TTestCase)
03 private
04   consulta: TConsultaCliente;
05 protected
06    procedure SetUp; override;
07    procedure TearDown; override;
08 published
09    procedure TestNomeEncontrado;
10    procedure TestNomeNaoEncontrado;
11 end;
12
13 implementation
14
15 { TTestCaseConsulta }
16
17 procedure TTestCaseConsulta.SetUp;
18 begin
19   inherited;
20   Application.CreateForm(Tdm, dm);
21   consulta := TConsultaCliente.Create;
22 end;
23
24 procedure TTestCaseConsulta.TearDown;
25 begin
26   inherited;
27   FreeAndNil(consulta);
28   FreeAndNil(dm);
29 end;
30
31 procedure TTestCaseConsulta.TestNomeEncontrado;
32 var
33   NomeABuscar: string;
34 begin
35   NomeABuscar := 'Barak Obama Hussein';
36
37   consulta.Consultar(NomeABuscar);
38   CheckTrue(dm.cdsClientesNome.AsString = NomeABuscar);
39 end;
40
41 procedure TTestCaseConsulta.TestNomeNaoEncontrado;
42 var
43   NomeABuscar: string;
44 begin
45   NomeABuscar := 'Bin Laden';
46   consulta.Consultar(NomeABuscar);
47
48   CheckFalse(dm.cdsClientesNome.AsString = NomeABuscar);
49   CheckTrue(dm.cdsClientesNome.AsString = 'Paulo Quicoli',
      'Não ficou
50     posicionado no primeiro registro');
51 end;
```

seja passada uma quantidade inferior, uma exceção do tipo *EParametroInvalidoEx* é levantada. É isso que vamos testar. Adicione um novo método *TestParametroInvalido* na classe *TTestCaseConsulta* e adicione uma referência à *Unit ExceptionsPersonalizadas*. Para realizar o teste vamos usar um método *check* especial, o *CheckException*. Esse método espera como parâmetro um tipo *TTestMethod* que na verdade é uma referência para métodos, conforme sua declaração a seguir.

```
TTestMethod  = procedure of object;
```

Isso significa que só podemos passar referências que obedeçam a assinatura definida em *TTestMethod*, uma *procedure* sem parâmetros. O segundo parâmetro é o tipo *exceção*, que em nosso caso deve ser *EParametroInvalidoEx*.

Lembre-se que o método que queremos testar é o *consulta.Consultar()*, que espera um parâmetro. Ele diverge da assinatura *TTestMethod* e não pode ser passado diretamente para o método *CheckException*. Então vamos encapsulá-lo em um método que tenha a assinatura esperada.

Resumo do DevMan

Em sua definição o padrão Adapter possui a tarefa de converter a interface de uma classe para outra interface que uma classe cliente espera, permitindo que classes trabalhem juntas, mesmo possuindo interfaces diferentes. O próprio Delphi aplica automaticamente o padrão Adapter para nós. Em outras palavras algumas vezes, uma classe projetada para ser reusada não o é porque sua interface não combina com a interface requisitada por uma aplicação de um domínio específico, um exemplo disso: uma classe TextView existente poderia ser reusada através de uma interface TextShape em um editor de desenhos para desenhar e organizar elementos gráficos (linhas, polígonos, etc.), tratando o texto como mais um tipo de elemento gráfico.

Nota: Para os mais observadores, vejam que estamos aplicando aqui um padrão de projeto, o *Adapter*.

Crie um método na seção *protected* chamado *ChamaConsulta* e sua implementação deve ser a seguinte:

```
procedure TTestCaseConsulta.ChamaConsulta;
begin
  consulta.Consultar('A');
end;
```

Em seguida implementamos o método *TestParametroInvalido* como a seguir:

```
procedure TTestCaseConsulta.TestParametroInvalido;
begin
  CheckException(ChamaConsulta,EParametroInvalidoEx);
end;
```

Execute os testes e veja que por enquanto está tudo *OK*.

Testando a entrada de dados

De forma similar aos testes da consulta vamos criar outra classe, conforme **Listagem 5**.

Listagem 5. Test Case de cadastro

```
01 TTestCaseCadastro = class(TTestCase)
02 private
03   cadastro: TCliente;
04 protected
05   procedure SetUp; override;
06   procedure TearDown; override;
07 published
08 end;
```

Observe que já preparamos aqui os métodos *Setup* e *TearDown*. Vamos implementá-los como na **Listagem 6**.

Como fizemos nos testes de consulta, aqui também centralizamos a criação e destruição do *DataModule* e do objeto de cadastro da camada de negócio. O primeiro teste que vamos realizar é o *Inserir*. Adicione um novo método chamado *TestInserir* à classe *TTestCaseCadastro* e o implemente como a seguir.

Listagem 6. Setup e TearDown

```
01 procedure TTestCaseCadastro.SetUp;
02 begin
03   inherited;
04   Application.CreateForm(Tdm, dm);
05   cadastro := TCliente.Create;
06 end;
07
08 procedure TTestCaseCadastro.TearDown;
09 begin
10   inherited;
11   FreeAndNil(cadastro);
12   FreeAndNil(dm);
13 end;
```

```
procedure TTestCaseCadastro.TestInserir;
begin
  cadastro.Inserir;
  CheckTrue(dm.cdsClientes.State in [dsInsert]);
end;
```

Veja que estamos verificando se após a chamada do método *Inserir* o estado do *cdsCliente* realmente está em modo de inserção. Para utilizarmos as constantes de estado do *Dataset* é preciso adicionar ao *uses* a *Unit DB*.

O próximo teste é o *Editar*. Criamos então um método *TestEditar* e verificamos o estado do *cdsCliente*, como o seguinte código.

```
procedure TTestCaseCadastro.TestEditar;
begin
  cadastro.Editar;
  CheckTrue(dm.cdsClientes.State in [dsEdit]);
end;
```

O teste do excluir já é um pouco diferente porque é preciso verificar se um registro foi excluído e se foi o registro correto que para nós é o primeiro que possui o nome de *Paulo Quicoli*. Crie o método *TestExcluir* e o implemente como na **Listagem 7**.

Listagem 7. Teste de exclusão

```
01 procedure TTestCaseCadastro.TestExcluir;
02 var
03   QtdeOriginalRegistros: integer;
04 begin
05   QtdeOriginalRegistros := dm.cdsClientes.RecordCount;
06   cadastro.Excluir;
07   CheckTrue(QtdeOriginalRegistros > dm.cdsClientes.RecordCount);
08   CheckTrue(dm.cdsClientesNome.AsString <> 'Paulo Quicoli','Excluir o
09             registro errado');
10 end;
```

Para saber se realmente foi removido um registro, verificamos o retorno do método *RecordCount* do *cdsCliente*, antes e depois da chamada do *Excluir*. Para saber o primeiro registro foi excluído mesmo, verificamos o nome atual. Bem simples. O *Salvar* também é um pouco diferente. Crie o método *TestSalvar* e o deixe como na **Listagem 8**.

Para verificar se um registro foi salvo, apenas estamos verificando o *RecordCount* do *ClientDataSet*. O método *Salvar* da camada de negócios também realiza algumas verificações. Por exemplo, precisamos verificar se os campos obrigatórios estão preenchidos e se o e-mail está no formato válido.

Fica restando apenas o método *Cancelar*. Faça como temos feito, crie um método *TestCancelar* e o codifique como a seguir:

Listagem 8. Testando o Salvar

```
01 procedure TTestCaseCadastro.TestSalvar;
02 var
03   QtdeOriginalRegistros: integer;
04 begin
05   QtdeOriginalRegistros := dm.cdsClientes.RecordCount;
06   dm.cdsClientes.Insert;
07   dm.cdsClientesNome.AsString := 'Teste';
08   dm.cdsClientesEmail.AsString := 'teste@email.com';
09   cadastro.Salvar;
10   CheckTrue(QtdeOriginalRegistros < dm.cdsClientes.RecordCount);
11
12   dm.cdsClientes.Insert;
13   dm.cdsClientesNome.AsString := '';
14   dm.cdsClientesEmail.AsString := 'teste@email.com';
15   CheckException(cadastro.Salvar, ECampoObrigatorioEx);
16
17   dm.cdsClientes.Insert;
18   dm.cdsClientesNome.AsString := 'Teste';
19   dm.cdsClientesEmail.AsString := '';
20   CheckException(cadastro.Salvar, ECampoObrigatorioEx);
21
22   dm.cdsClientes.Insert;
23   dm.cdsClientesNome.AsString := 'Teste';
24   dm.cdsClientesEmail.AsString := 'testeemail.com';
25   CheckException(cadastro.Salvar, EFormatoInvalidoEx);
26 end;
```

```
procedure TTestCaseCadastro.TestCancelar;
begin
  dm.cdsClientes.Insert;
  cadastro.Cancelar;
  CheckTrue(dm.cdsClientes.State in [dsBrowse]);
end;
```

Aqui verificamos novamente apenas o estado do *cdsClientes*. Para rodar os testes do cadastro também é preciso registrá-lo na seção *Initialization* como o seguinte código:

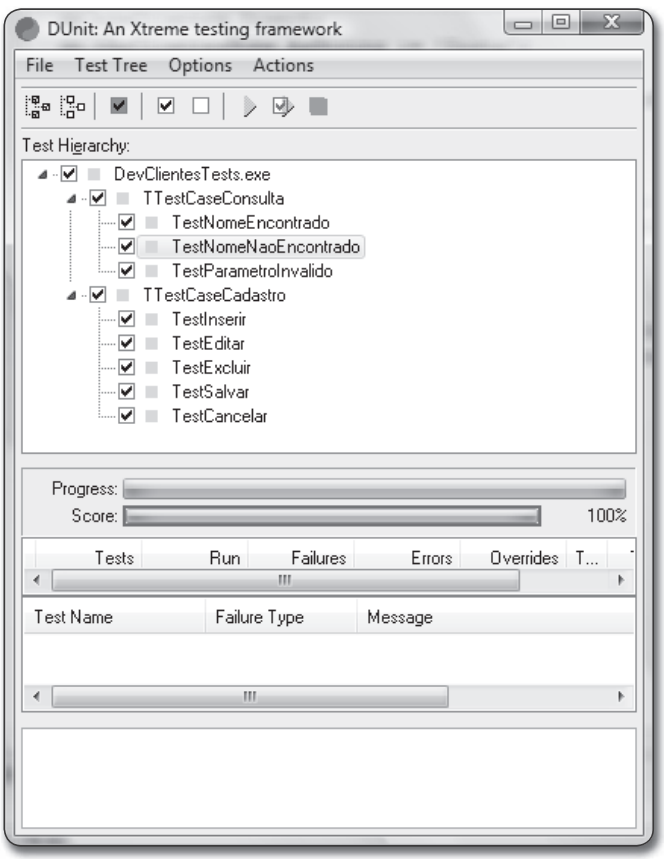


Figura 6. A execução de todos os testes

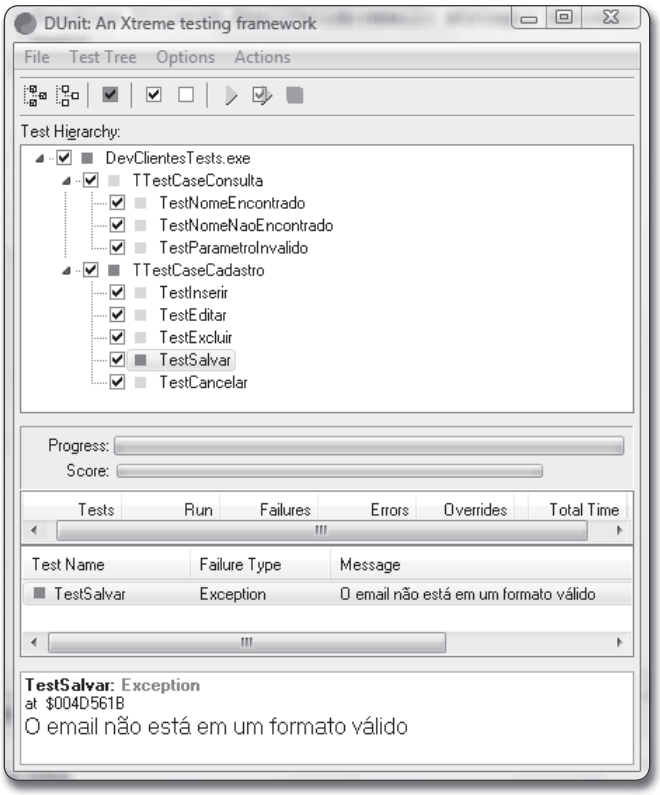


Figura 7. Teste falho

```
initialization
  TestFramework.RegisterTest(TTestCaseConsulta.Suite);
  TestFramework.RegisterTest(TTestCaseCadastro.Suite);
```

Execute os testes, o resultado será com o da **Figura 6**.

Fazendo um teste falhar

Até o momento todos os testes passaram, vamos simular agora uma falha no sistema. Digamos que fizemos algumas alterações e *sem querer* mudamos o código que valida o formato do *e-mail* durante o cadastro de cliente. Essa validação está na *Unit NegocioU.pas*, na classe *TCliente*, método *EmailValido*. Originalmente seu código é este:

```
result := Pos('@',email) > 0;
```

Vamos alterá-lo para:

```
result := Pos('?',email) > 0;
```

E então execute novamente os testes. O resultado está na **Figura 7**.

Ao detectar o erro podemos facilmente abrir o teste e averiguar de forma pontual o que está acontecendo e então corrigir sem demoras.

Conclusão

Acredito que todos perceberam a importância dos testes unitários em um sistema. Neste pequeno exemplo cobrimos quase que na totalidade a verificação de falhas, imagine em um sistema real como isso não seria bom.

Contudo para se ter os benefícios desses testes é preciso investir em arquitetura. É muito comum os projetos começarem com o famoso código *espaguete* e continuarem assim por todo sempre, impedindo a utilização de boas práticas. Meu conselho é que leiam, estudem, pesquisem sobre arquitetura de software e tentem aplicar no dia-a-dia, mesmo que o material encontrado seja para outra linguagem isso não quer dizer que você não pode aplicar em *Delphi*. Até mais. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos sobre técnicas que poderão aumentar a qualidade do desenvolvimento de software

Parser de expressões lógico-matemáticas

Veja como construir um editor de fórmulas no Delphi

Nesse artigo veremos

- Criação de um parser para criação de um editor de fórmulas;
- Boas práticas de desenvolvimento.

Qual a finalidade

Criar um editor de fórmulas para evitar que o usuário torne-se escravo do programador.

Quais situações utilizam esse recurso?

Em qualquer sistema que faz uso de fórmulas matemáticas, tais como cálculos de impostos, fretes, pedágios ou mesmo notas de um aluno;

Resumo do DevMan

Certamente o que um desenvolvedor menos gosta de fazer em um sistema é dar manutenção nele. Isso por uma série de fatores, tais como: custos envolvidos, horas de programação, atrasos em outras tarefas, alocação de recursos que poderiam estar trabalhando em outras tasks e etc.

A grande maioria dos sistemas é constituída de telas de cadastro, relatórios e regras de negócio. Dependendo do ramo de atividade, um sistema pode funcionar com pequenas fórmulas que fazem parte das regras de negócio, ou seja, os resultados de uma conta ou fórmula podem determinar o fluxo de uma aplicação. Dar manutenção nisso pode ser trabalhoso e oneroso para a empresa e os profissionais envolvidos.

Por que não então, desenvolver um editor de fórmulas para que o próprio usuário monte suas regras de negócios? Claro que esse é um tema bastante complexo, mas conseguiremos ver como montar tal editor.

Elvis Fusco

fusco@univem.edu.br

É mestre em Ciência da Computação pelo UNIVEM de Marília e Doutorando em Ciência da Informação pela UNESP de Marília. Gerente do Departamento de Sistemas e Coordenador dos cursos de Ciência da Computação e Sistemas de Informação do UNIVEM de Marília. Pode ser contatado pelo endereço fusco@univem.edu.br.

Uma das qualidades que um sistema de informação deve ter é a flexibilidade em permitir ao usuário customizar o sistema de acordo com as regras de negócios da empresa. Esse conceito também, às vezes chamado de parametrização, possibilita que se o domínio de aplicação se altera, o usuário não necessita chamar o desenvolvedor para alterar as regras no sistema, ele mesmo pode fazer essas alterações no próprio software. Isso é uma boa prática de desenvolvimento de sistemas de informação.

Em sistemas com regras baseadas em fórmulas, uma maneira de implementar esse recurso é por meio de *parsers* de expressão para funções *lógico-matemáticas*. Esses *parsers* analisam, validam e executam expressões matemáticas e lógicas que utilizam-se de parâmetros vindos do sistema de informação e operadores relacionais, lógicos e matemáticos.

Podem ser utilizados, nesses *parsers*, os operadores relacionais (*comparação*), lógicos, fatoriais, atribuição de valores, parâmetros definidos pelo usuário e funções lógicas além de variáveis definidas pelo desenvolvedor.

Na prática, podemos utilizar esse recurso criando em nossos sistemas interfaces em que o usuário monta fórmulas de cálculo para regras de domínio de uma aplicação. Por exemplo, para cálculo de salários de um funcionário em um sistema de folha de pagamento, definição das regras de média e aprovação em sistemas de gestão escolar e cálculo de geração de valores de pagamento em sistemas financeiros.

Para demonstrar a utilização desse recurso em sistemas de informação, vamos criar nesse artigo uma aplicação para definição, por parte do usuário, de fórmulas de cálculo de nota final e resultado em um sistema de gestão escolar. Para tanto iremos criar uma interface em que o usuário possa definir os parâmetros que formam a nota e a situação final das disciplinas, suas fórmulas de cálculo e validar essas expressões.

Componente TExpressionParser

Para implementar a aplicação iremos utilizar o componente *freeware TExpressionParser* que é um *parser* flexível de expressões para funções *lógico-matemáticas*. Na **Tabela 1** são mostrados os operadores que podem ser utilizados nas expressões. E na **Tabela 2** são mostradas as funções que podem ser utilizadas nas expressões.

Tipo de Operação	Operadores
Operações matemáticas	+ - * / ^ div mod
Comparações relacionais (também para strings)	> < <> = <= >=
Operações lógicas	and or xor not
Fatoriais	!
Percentagem	%
Atribuição a variáveis	:=

Tabela 1. Operadores do componente TExpressionParser

Função	Descrição
Abs(x)	Retorna o valor absoluto de um número
Max(x;y)	Retorna o maior valor entre 2 números
Min(x;y)	Retorna o menor valor entre 2 números
Round(x)	Arredonda um número real
If(expressão; valor_se_verdadeiro; valor_se_falso)	Retorna um valor dependendo do resultado (verdadeiro ou falso) de uma expressão

Tabela 2. Funções do componente TExpressionParser

Nota do DevMan

Um Parser é um programa de computador (ou apenas um componente de um programa) que serve para analisar a estrutura gramatical de uma entrada, manipulando os tokens, que são segmentos de texto ou símbolos que podem ser manipulados. Em XML, por exemplo, o parser pode ser um leitor que ajuda na conversão do arquivo para manipulação dos dados contidos no mesmo.

Nota: O componente *TExpressionParser* está disponível para download gratuito no site do desenvolvedor em *www.aew.wur.nl/UK/TExpressionParser*, onde você encontra também a documentação e os fontes do componente.

Criando a Aplicação

Vamos criar uma aplicação que permita ao usuário criar fórmulas de cálculo de notas finais e resultado para sistemas de gestão acadêmica. Na **Figura 1**, pode ser vista a tela que será implementada. Veja que nossa tela é bastante complexa, tem muitos botões e caixas para seleção de informações. Mas não se preocupe, porque não é tão complexo o código-fonte assim.

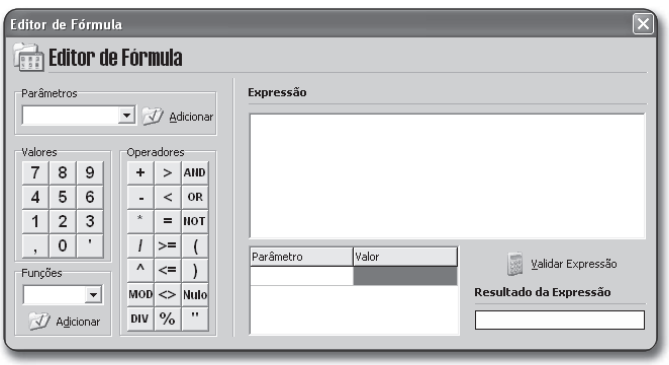


Figura 1. Editor de Fórmula

Abra o *Delphi* e crie uma nova aplicação usando *File>New Application* e salve-a como *PEditorFormula.dpr* e a *Unit* do formulário principal como *UEditorFormula.pas*. Vamos fazer a definição dos parâmetros da aplicação. Na parte esquerda temos alguns *GroupBoxes* e dentro deles alguns *SpeedButtons* e *ComboBoxes*. Em especial, inclua alguns itens na propriedade *Items* do primeiro *ComboBox*(*cmbParametros)* como na **Figura 2**.

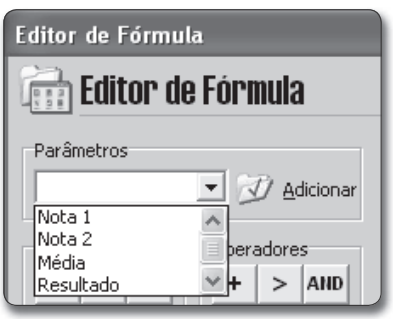


Figura 2. Parâmetros utilizados nas fórmulas

Em uma aplicação real, poderíamos trazer essas informações diretamente do banco de dados. Assim ficaria mais fácil fazer a inclusão de novas fórmulas.

O componente *TExpressionParser* permite a utilização de algumas funções pré-definidas como vimos anteriormente. Vamos criar um *ComboBox* (*cmbFuncoes*) com essas funções, de modo que o usuário possa escolher entre uma delas adicioná-las ao editor de fórmulas usando um *SpeedButton* (*spbAdicionarFuncao*) conforme a **Figura 3**. Veja que nosso sistema começa a tomar forma.

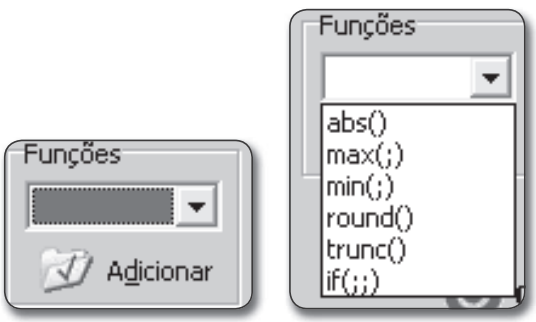


Figura 3. Funções pré-definidas

No lado direito da tela encontramos um componente *Memo* onde faremos a digitação das fórmulas em si (**Figura 4**).

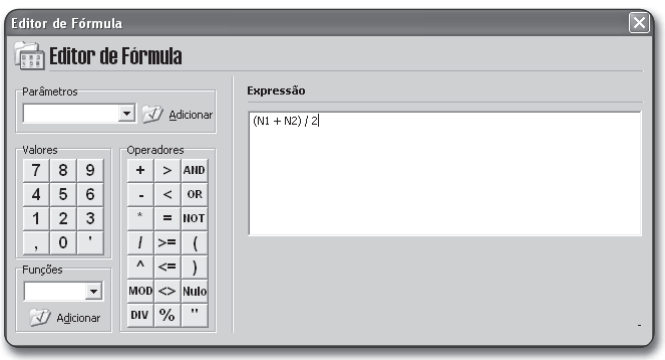


Figura 4. Editor de fórmula com um Memo para definição das expressões

Inserindo valores nos parâmetros

Ao criar a fórmula, antes de colocá-la em produção, o usuário deve testar o funcionamento lógico da mesma, para tanto, é necessário que ele coloque valores nos parâmetros das fórmulas e o *parser* faça a validação da expressão. Neste contexto colocaremos um componente do tipo *ValueListEditor*(*ValueListEditor1*) alterando as propriedades *KeyOptions>KeyEdit* para *True* e *TitleCaptions* para *Parâmetro | Valor*.

Para finalizar a interface da aplicação coloque um *SpeedButton* (*spbValidar*) para executar a validação da expressão e um *Edit* (*edtResultado*) para mostrar o resultado da expressão (**Figura 5**). A explicação é bem simples: vamos possibilitar que o nosso usuário final possa montar a fórmula e testá-la antes de salvar para um banco de dados por exemplo.

Nota: O componente *ValueListEditor* é semelhante, grosseiramente falando, ao *Property Editor* do Delphi. Note que é possível incluirmos uma propriedade, nesse caso apelidamos de *parâmetro*, e vinculamos a ela um valor (*Value*).

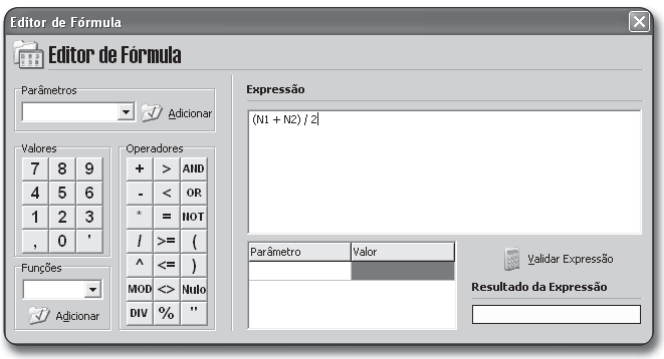


Figura 5. Interface do editor de fórmula

Adicionando os parâmetros

Os parâmetros em nossa aplicação serão os personagens principais para construção das fórmulas. Então o que faremos é adicionar o parâmetro em nosso *ValueListEditor* juntamente com seu valor para posteriormente ser avaliado pelo construtor de fórmulas. Vamos começar inserindo o código que fará a inclusão do parâmetro ao componente mencionado. No evento *OnClick* do botão *spbAdicionarParametros* que adiciona ao *Memo* o parâmetro escolhido e também ao controle *ValueListEditor* insira o código a seguir:

```
if cmbParametros.Text <> '' then
begin
  AddToMemo(cmbParametros.Text);
  ValueListEditor1.InsertRow(cmbParametros.Text, '', True);
end;
```

Veja que é bem simples. Estamos verificando se o conteúdo do *ComboBox* de parâmetros é diferente de vazio e então chamamos o método *AddToMemo* informando qual o parâmetro deve ser inserido. Na seqüência incluimos os valores no componente *ValueListEditor* através do seu método *InsertRow,* que por sua vez insere uma nova linha ao controle. Em outras palavras uma nova propriedade ou parâmetro.

Foi utilizada a função *AddToMemo* para posicionar o conteúdo na posição do cursor dentro do *Memo*. Você deve criar essa função como *private* no seu formulário e implementá-la como na **Listagem 1**. Perceba que estamos trabalhando o tempo todo com o método *Pos* do Delphi para analisar a *string* da fórmula e descobrir onde estão os caracteres especiais, como parênteses por exemplo.

Adicionando os valores e operadores

Quando o usuário clicar sobre os botões referentes aos valores, constantes e operadores, o valor da propriedade *Caption* desses componentes deve ser adicionado ao editor. Isso é bem fácil de ser feito. Apenas chamamos o método *AddToMemo,* já comentado anteriormente, passando para ele o *Caption* do componente.

Listagem 1. Função AddToMemo

```
01 procedure TfrmEditorFormula.AddToMemo(S: string);
02 var
03   Len: Integer;
04   IsFunction: Boolean;
05 begin
06   if S = '' then
07     S := ' ';
08   with memFormula do
09   begin
10     IsFunction := Pos('(', S) > 0;
11     Len := Sellength;
12     if IsFunction and (Len > 0) then
13     begin
14       Sellength := 0;
15       SelText := Copy(S, 1, Pos('(', S));
16       SelStart := SelStart + Len;
17       SelText := Copy(S, Pos('(', S) + 1, Len); ;
18       SelStart := SelStart - 1;
19     end
20     else
21     begin
22       SelText := S;
23       if IsFunction then
24         SelStart := SelStart - 1;
25     end;
26   end;
27 end;
```

Como não sabemos ao certo qual componente chamou a função, então montamos um *typecast* com o *Sender*, pois dessa forma conseguimos identificar facilmente o *Caption* do componente que está enviando a operação. Programe o evento *OnClick* do botão *btn7* de acordo com o seguinte:

```
AddToMemo((Sender as TButton).Caption);
```

Nesse evento está sendo adicionado o *Caption* do botão *btn7*. Nos outros botões referencie o evento *OnClick* a esse evento conforme a **Figura 6**.

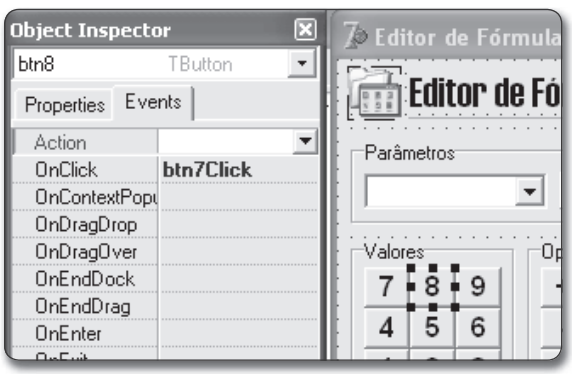


Figura 6. Referenciando o evento btn7Click

Adicionando as funções

As funções também são parte importantíssima em nossa aplicação. Faremos com que seja possível utilizar as mais variadas funções para montagem de nossas fórmulas. No entanto, apenas estamos adicionando ao *memo* o conteúdo presente na propriedade *Text* do *cmdFuncoes*, ou seja, o *cliente/usuário* deve ficar atento quando for utilizar-se de funções. Insira então o código-fonte da linha a seguir no evento *OnClick* do botão *spbAdicionarFuncao*.

```
if cmbFuncoes.Text <> '' then
  AddToMemo(cmbFuncoes.Text);
```

Validando a expressão

Nesse momento faremos a validação do que programamos até aqui. Finalmente vamos criar a função que fará o *parser* de validação e, utilizando o componente *TExpressionParser*, processará a expressão retornando o valor resultante da fórmula.

Para utilizar esse componente em nossa aplicação:

- Adicione na pasta do projeto os arquivos do componente: *ParseExpr.pas* e *ParseClass.pas*.
- Adicione na cláusula *Uses*, a classe *ParseExpr*.

Depois criamos a função *ResultadoExpressao* conforme **Listagem 2**.

Listagem 2. Função ResultadoExpressao

```
01 function TfrmEditorFormula.ResultadoExpressao(Formula:
    string) : string;
02 var
03   MyParser: TExpressionParser;
04   Comando, ComandoOK, P1, Posicao: string;
05   I, PosIni: Integer;
06   VarStr : TStrings;
07 begin
08   try
09     MyParser := TExpressionParser.Create;
10     MyParser.ClearExpressions;
11     Comando := AnsiUpperCase(Formula);
12     P1:= Comando;
13     VarStr := TStringList.Create;
14     PosIni := -1;
15     Posicao:='1';
16     ComandoOK:='';
17     for i:= 1 to Length(P1)  do
18     begin
19      if P1[i] = '"' then // x = (30 - 40) * 10
20      begin
21         if PosIni = -1 then
22           PosIni := i
23         else
24         begin
25           VarStr.Add(Copy(P1, PosIni, i - PosIni+1));
26           ComandoOK := ComandoOK + Posicao;
27           Posicao:= IntToStr(StrToInt(Posicao)+1);
28           PosIni := -1;
29         end
30       end
31       else if PosIni = -1 then
32         ComandoOK := ComandoOK + P1[i];
33     end;
34     Comando:= ComandoOK;
35     for i := 0 to ValueListEditor1.Strings.Count -1 do
36     begin
37       if ValueListEditor1.Strings.Values[
38         AnsiUpperCase(ValueListEditor1.Strings.Names[i])] = '' then
39         Comando := StringReplace(Comando, AnsiUpperCase(
40           ValueListEditor1.Strings.Names[i]), '-1' ,
              [rfReplaceAll])
41       else
42         Comando := StringReplace(Comando,ANSIUPPERCASE(
43            ValueListEditor1.Strings.Names[i]),
44           AnsiUpperCase(ValueListEditor1.Strings.Values[
45           AnsiUpperCase(ValueListEditor1.Strings.Names[i])]),
              [rfReplaceAll]);
46     end;
47    Comando := StringReplace(Comando,'NULO','-1',[rfReplaceAll]);
48     MyParser.Optimize := TRUE;
49     I := MyParser.AddExpression(comando);
50     if VarStr.Count > 0 then
51       Result := VarStr.Strings[StrToInt(MyParser.AsString[I])-1]
52     else
53       Result := MyParser.AsString[I];
54   finally
55     MyParser.Free;
56   end;
57 end;
```

Criamos uma instância do componente *TExpressionParser* para que possamos usá-la. Em seguida limpamos as expressões de sua memória apenas por precaução, já que acabamos de criá-lo. Isso é feito chamando o método *ClearExpressions* (*Linha 10*). Na

seqüência usamos o método *AnsiUpperCase* para transformar em maiúsculas os caracteres que compõem nossa fórmula, isso evita que o componente se confunda entre maiúsculas ou minúsculas na hora de executar o comando da fórmula e também para criar um padrão. Repare que passamos para o método a variável *Formula* que na verdade é o parâmetro recebido pela função. O resultado do método é atribuído a variável *Comando*.

Utilizamos uma nova variável (*P1*) que será usada para avaliar o conteúdo da fórmula. Percorremos *P1* usando um laço *for*. Na realidade nosso *for* faz uma varredura no *P1* e adiciona os comandos encontrados em uma lista de *strings* (*VarStr = TStrings*) (*Linhas 17 a 33*). Caso o caractere seja uma aspa dupla, analisamos se a posição inicial (*PosIni*) é -1 (primeira volta no laço *for*) então ignoramos e atribuímos *I* ao *PosIni*. Caso contrário, adicionamos ao *VarStr* (*nossa lista de strings*) o caractere atual levando em conta a posição inicial + 1. Ainda concatenamos o conteúdo de ComandoOk com a Posição

Por fim fazemos um novo laço *for*, dessa vez em nosso controle *ValueListEditor* em busca das variáveis armazenadas e então alimentamos a variável *Comando* com as respectivas, ou seja, caso o usuário tenha incluído alguma variável na fórmula, a mesma é procurada dentro do *ValueListEditor* e acrescentada à fórmula final. Usamos o método *StringReplace* para fazer isso. Então chamamos o método *Optimize* do componente que ativa um aprimoramento no algoritmo interno e adicionamos o comando (expressão) à sua lista de expressões usando o método *AddExpression*.

Em resumo, nessa função passamos a fórmula contida no *Memo* e ele (o componente) devolve o valor resultante da expressão. Para tanto, antes de executar o *parser*, os parâmetros são substituídos pelos valores informados pelo usuário ou, em uma aplicação real, pelos valores vindos de um banco de dados.

Para finalizar, implemente o evento *OnClick* do *spbValidar* que invoca a função *ResultadoExpressao* passando como parâmetro o conteúdo do *Memo*. Veja que teremos que criar a *function ResultadoExpressao* que vimos anteriormente na **Listagem 2**.

```
edtResultado.Text := ResultadoExpressao(memFormula.Text);
```

Agora vamos validar a aplicação criando uma fórmula de cálculo de média e depois de resultado da disciplina conforme as **Figuras 7** e **8**. Perceba que não implementamos tanto código assim, mas conseguimos fazer com que o sistema comporte-se como você desejar.

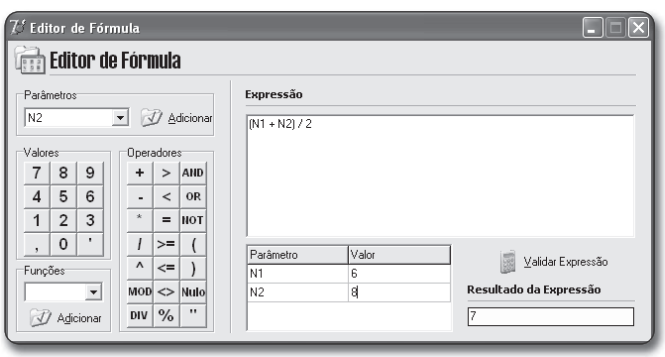


Figura 7. Fórmula de média de notas

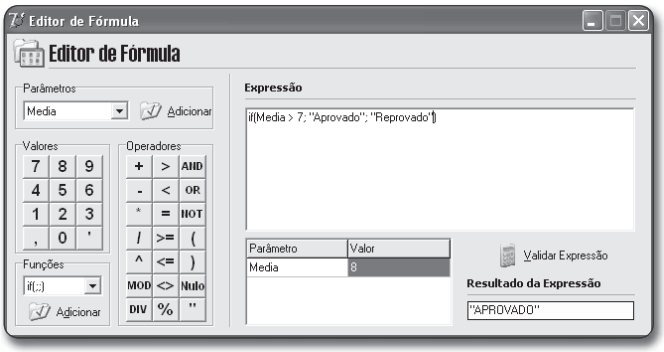


Figura 8. Fórmula de resultado do aluno

Colocando em prática

Até agora vimos uma forma de construir as fórmulas. Mas como vamos utilizá-las? É isso que veremos agora. Faremos um pequeno projeto que calcula a média de notas de alguns alunos. Vamos calcular essas médias utilizando a fórmula aritmética e ponderada, e essas fórmulas serão executadas pelo componente.

Inicie um novo projeto no *Delphi*, salve-o como *ExemploReal*. Adicione agora um *DataModule* e nele vamos adicionar três *ClientDataSets*. No primeiro adicione os seguintes campos: *peso1*, *peso2*, *peso3* e *peso4*, todos inteiros. Altere a propriedade *Name* desse *ClientDataSet* para *cdsParametros*. No segundo vamos alterar seu *name* para *cdsFormulas* e adicionar os seguintes campos: *Titulo (string de 20)*, *Descricao (string de 50)* e *Formula (string de 100)*. No terceiro e último *ClientDataSet* vamos adicionar os campos: *Nome (string de 50)*, *Nota1*, *Nota2*, *Nota3*, *Nota4* do tipo inteiro e *Media* do tipo *Float*. Por fim, adicione três componentes *TDataSource* e ligue cada um com um *ClientDataSet*.

Vamos agora criar três *procedures* no *DataModule* que irão inserir alguns registros nesses *ClientDataSet*s. Na **Listagem 3** temos a implementação dessas *procedures*. Não há segredos, apenas criamos um *DataSet* e utilizamos os métodos *Insert* e *Post* do *ClientDataSet* para inserir dados fictícios nas tabelas. Nosso formulário deverá se parecer com a **Figura 9**.

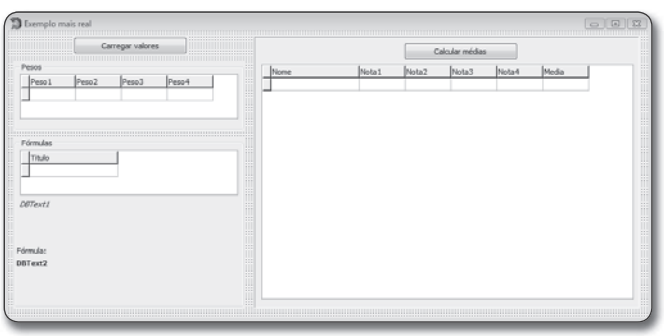


Figura 9. Formulário principal

Observe na **Figura 9** que ligamos os *DBGrids* aos seus respectivos *DataSources*. No evento *onClick* do botão *Carregar valores* vamos executar as *procedures* que irão preencher os dados, como é visto no código a seguir:

Listagem 3. Procedures que irão preencher os ClientDataSets

```
01 procedure Tdm.CarregaAlunos;
02 begin
03   cdsAlunos.CreateDataSet;
04   cdsAlunos.Insert;
05   cdsAlunosNome.AsString := 'Aluno 1';
06   cdsAlunosNota1.AsInteger := 6;
07   cdsAlunosNota2.AsInteger := 7;
08   cdsAlunosNota3.AsInteger := 5;
09   cdsAlunosNota4.AsInteger := 5;
10   cdsAlunos.Post;
11   cdsAlunos.Insert;
12   cdsAlunosNome.AsString := 'Aluno 2';
13   cdsAlunosNota1.AsInteger := 5;
14   cdsAlunosNota2.AsInteger := 9;
15   cdsAlunosNota3.AsInteger := 6;
16   cdsAlunosNota4.AsInteger := 8;
17   cdsAlunos.Post;
17   cdsAlunos.Insert;
18   cdsAlunosNome.AsString := 'Aluno 3';
19   cdsAlunosNota1.AsInteger := 8;
20   cdsAlunosNota2.AsInteger := 9;
21   cdsAlunosNota3.AsInteger := 7;
22   cdsAlunosNota4.AsInteger := 3;
23   cdsAlunos.Post;
24 end;
25 procedure Tdm.CarregaFormulas;
26 begin
27   cdsFormulas.CreateDataSet;
28   cdsFormulas.Insert;
29   cdsFormulasTitulo.AsString := 'Média aritmética';
30   cdsFormulasDescricao.AsString := 'Esta média é obtida por ' +
31                                    'somar as 4 notas e o
                                       resultado ' +
32                                    'obtido é divido por 4';
33   cdsFormulasFormula.AsString :='(Nota1+Nota2+Nota3+Nota4)/4';
34   cdsFormulas.Post;
35
36   cdsFormulas.Insert;
37   cdsFormulasTitulo.AsString := 'Média ponderada';
38   cdsFormulasDescricao.AsString := 'Esta média é obtida por ' +
39                                    'multiplicar cada nota ao
                                       seu peso e então '+
40                                    'somar o resultado. Depois
                                       esse resultado ' +
41                                    'da soma é dividido pela
                                       soma dos 4 pesos';
42   cdsFormulasFormula.AsString :=
43 '((Nota1*Peso1)+(Nota2*Peso2)+(Nota3*Peso3)+(Nota4*Peso4))/' +
44                                    '(Peso1+Peso2+Peso3+Peso4)';
45   cdsFormulas.Post;
46 end;
47 procedure Tdm.CarregaPesos;
48 begin
49   cdsParametros.CreateDataSet;
50   cdsParametros.Insert;
51   cdsParametrosPeso1.AsInteger := 3;
52   cdsParametrosPeso2.AsInteger := 2;
53   cdsParametrosPeso3.AsInteger := 1;
54   cdsParametrosPeso4.AsInteger := 4;
55   cdsParametros.Post;
56 end;
```

Listagem 4. Calculando as médias

```
01 procedure TPrincF.Button1Click(Sender: TObject);
02 var
03  media: string;
04 begin
05   dm.cdsAlunos.First;
06   while not dm.cdsAlunos.Eof do
07   begin
08     media := ExecutarFormula(dm.cdsFormulasFormula.AsString);
09     dm.cdsAlunos.Edit;
10     dm.cdsAlunosMedia.AsString := media;
11     dm.cdsAlunos.Post;
12     dm.cdsAlunos.Next;
13   end;
14 end;
```

```
dm.CarregaFormulas;
dm.CarregaPesos;
dm.CarregaAlunos;
btCriar.Enabled := false;
```

E no botão *Calcular médias* realizamos um *loop* do *ClientDataSet* que contém os dados dos alunos e a cada registro calculamos a média pela função *ExecutarFormula*, como é mostrados na **Listagem 4**.

Observem que na *linha 8* da **Listagem 4** o valor da média é retornado pela função *ExecutarFormula* e que posteriormente apenas editamos o registro atual e gravamos nele o valor obtido. Na **Listagem 5** vemos como é implementada a função *ExecutarFormula*.

Na *Linha 11* realizamos uma substituição na fórmula. Substituímos as variáveis *PESO1, PESO2, PESO3* e *PESO4* pelos seus respectivos valores constantes no *ClientDataSet Parametros*. Na *Linha 16* realizamos o mesmo, só que substituímos as variáveis *NOTA1, NOTA2, NOTA3* e *NOTA4* pelos respectivos valores do registro atual no *ClientDataSet cdsAlunos*. Execute a aplicação e veja o resultado.

Conclusão

Em sistemas de informações são muitas as situações em que podemos utilizar os recursos vistos neste artigo de modo que permitamos que as regras de negócio de uma aplicação sejam configuradas de um modo parametrizado na aplicação

Esta é uma boa prática de desenvolvimento de software, pois o domínio de uma aplicação é dinâmico e quanto mais flexível for seu sistema maior será a facilidade de manutenção, adaptação e aderência à realidade do cenário em que ele será utilizado. ●

Listagem 5. Função ExecutarFormula

```
01 function TPrincF.ExecutarFormula(Formula: string): string;
02 var
03  s: string;
04  x: integer;
05  MyParser: TExpressionParser;
06  i: integer;
07 begin
08  s := AnsiUpperCase(Formula);
09  //Parse dos pesos
10  for x:= 1 to 4 do
11    s := stringReplace(s,'PESO'+IntToStr(x),
12  dm.cdsParametros.FieldByName('Peso'+IntToStr(x)).
     AsString,[rfReplaceAll]);
13  //Parse das notas
14  for x:= 1 to 4 do
15    s :=
16  stringReplace(s,'NOTA'+IntToStr(x),
17  dm.cdsAlunos.FieldByName('Nota'+IntToStr(x)).
     AsString,[rfReplaceAll]);
18  MyParser := TExpressionParser.Create;
19  MyParser.ClearExpressions;
20  MyParser.Optimize := TRUE;
21  I := MyParser.AddExpression(s);
22  Result := MyParser.AsString[I];
23  FreeAndNil(MyParser);
24 end;
```

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos para iniciantes na linguagem Delphi

Google Maps com Win32

Integre seus sistemas com o Google Maps

Nesse artigo veremos
- Utilização de mapas em aplicações Win32;
- Criando uma apontador de rotas.

Qual a finalidade
Entender o funcionamento do Google Maps e aplicar a soluções Win32;

Quais situações utilizam esse recurso?
A aplicação de mapas em aplicações de todo tipo tem se tornado um padrão e uma mania nacional. Nesse artigo veremos como utilizar a API do Google Maps em nossas aplicações Win32;

Adriano Santos
falecom@adrianosantos.pro.br
É desenvolvedor Delphi desde 1998, professor e programador PHP, bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É Editor Chefe das revistas ClubeDelphi e WebMobile. Gerente de TI na SoftPark, parceira Borland, membro fundador do DUG-SP (www.dug-sp.com) e mantenedor do blog Delphi to Delphi (www.delphito-delphi.blogspot.com) com dicas, informações e tudo sobre o desenvolvimento Delphi.

Já há algum tempo venho acompanhado de perto a evolução de uma das empresas que mais admiro no mercado de TI: o *Google*. Suas idéias são sempre inovadoras e ousadas e acabam se tornando referência no mercado. Aqui mesmo na *DevMedia* e no meu dia-a-dia utilizo algumas ferramentas, tais como *Google Docs* (*Editor de Textos*) e *Google Spreadsheats* (*Planilha Eletrônica*), além é claro do *Orkut* e *GMail*. Isso não é uma propaganda e sim a divulgação gratuita de recursos que realmente podem nos ajudar no dia-a-dia. Uma ferramenta me chamou bastante a atenção em particular nos últimos dias: o *Google Maps*.

Resumo do DevMan

Grande parte dos Web Sites de empresas, bares e instituições no geral, oferecem o recurso de mapa para localização do estabelecimento. Porém, há muito mais formas de utilização de mapas em nossas aplicações. Imagine por exemplo que nosso usuário possa ver um mapa com a localização exata de seu cliente ao abrir o cadastro dele. Ou então que o sistema possa traçar uma rota para que o carro de entrega da empresa possa saber como chegar ao seu destino.

Tudo isso é possível ser aplicado em sistemas Win32, o que é bem diferente do uso na internet. Veremos nesse artigo como fazer isso e como funcionam os mapas do Google.

Precisei incluir mapas em minhas aplicações *Win32* e *Intraweb* e por esse motivo acabei tendo que estudar sua *API*. Acabei descobrindo que podemos utilizá-lo das mais variadas formas, inclusive em aplicações *Win32*.

O que acham de mostrarmos um mapa com a localização exata de um cliente ao consultar seu cadastro? O mais interessante é tudo isso é gratuito e está disponível na internet para usarmos e abusarmos. Vejamos então nesse artigo como fazer uso dessa tecnologia em aplicações *Win32*.

Adquirindo uma licença de uso

Isso mesmo, embora seja gratuito o *Google Inc* exige que façamos um registro para obter a facilidade de utilização de seus produtos, mas não se assuste, pois não receberá nenhum boleto em sua residência e nem é tão difícil assim fazer o registro. Mão na massa!

A primeira coisa é acessarmos o site do *Google* que refere-se ao *Google Maps*. Acesse então o link *http://code.google.com/apis/maps/*, onde é possível adquirir todo tipo de informação a respeito da *API* do *Google*. À direita do site encontramos o link *Sign up for a Google Maps API key* e é esse link que contém as instruções e onde podemos obter a licença de uso. Ao final do contrato encontramos um campo para preenchimento. Devemos informar nesse campo, qual o link do nosso site que utilizará o recurso. Podemos informar tranquilamente quantos links quisermos. Ao pressionar o botão *Generate API Key*, uma *string* será criada e algumas instruções são dadas, veja **Figura 1**.

Essa *string* será usada sempre que for necessário fazer alguma consulta. Veremos mais adiante. O que vemos nos dois quadros em seguida são para qual link essa *string* se refere e um pequeno exemplo de uso. Veja na **Figura 2** o local exato onde o código gerado pelo *Google* é utilizado. O que acontece é que todo código gerado para utilização dos mapas, precisa ter esse número de licenciamento. Perceba que o código é *HTML* e *JavaScript*, o que nos dará um pouco de trabalho ao implementá-lo em *Win32*.

Entendendo o uso do Google Maps

Há realmente uma vasta enciclopédia no link *exemplos de API's do Google Maps* encontrado a direita desse site. É possível lermos e baixarmos os mais variados tipos de mapas e consultas. Para usarmos o *Google Maps* precisamos criar uma página *HTML* e nela colocar o código *JavaScript* necessário para a consulta. Ele ficará entre as tags *<head>* e *</head>*. Para quem já desenvolve ou desenvolveu alguma página *HTML* isso é fácil, mas para que nunca colocou a mão na massa pode parecer um bicho de 7 cabeças. Posso afirmar com ganho de causa que é bastante simples a utilização. Vamos analisar primeiramente um mapa bem simples.

Acesse o link *http://code.google.com/intl/pt-BR/apis/maps/documentation/examples/*, onde encontramos uma série de exemplos do *Google Maps*. Clique com o botão direito sobre o *link map-simple.html* e escolha a opção *Salvar esse link como*, assim teremos o código *HTML* desse arquivo. Seu código-fonte está descrito na **Listagem 1**. Esse código gera o mapa visto na **Figura 3**.

```
ABQIAAAATD0hHGY27tp5XsLI5uH-0BQdk7bZ5PREDyjqNWHotX1GHS13oxSb5c8B7ufX121cShasXxkfhhJX6g
```

Figura 1. String gerada pelo Google Maps API

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8"/>
    <title>Google Maps JavaScript API Example</title>
    <script src="http://maps.google.com/maps?file=api&amp;v=2&amp;key=ABQIAAAATD0hHGY27tp5XsLI5uH-
0BQdk7bZ5PREDyjqNWHotX1GHS13oxSb5c8B7ufX121cShasXxkfhhJX6g"
      type="text/javascript"></script>
    <script type="text/javascript">

    //<![CDATA[

    function load() {
      if (GBrowserIsCompatible()) {
        var map = new GMap2(document.getElementById("map"));
        map.setCenter(new GLatLng(37.4419, -122.1419), 13);
      }
    }

    //]]>
    </script>
  </head>
  <body onload="load()" onunload="GUnload()">
    <div id="map" style="width:500px;height:300px"></div>
  </body>
</html>
```

Figura 2. Utilização da string de licença

Listagem 1. Código simples de página com Google Maps

```
01 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
02   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
03 <html xmlns="http://www.w3.org/1999/xhtml"
04   xmlns:v="urn:schemas-microsoft-com:vml">
05   <head>
06     <meta http-equiv="content-type" content="text/html;
         charset=utf-8"/>
07     <title>Google Maps JavaScript API Example: Simple Map</
         title>
08     <script src="http://maps.google.com/maps?file=api&amp;v=2&amp;
09       sensor=false&amp;
10       key=ABQIAAAAzr2EBOXUKnm_jVnkOOJI7xSosDVG8KKPE1-m51RBrvYughuyMxQ-
11         i1QfUnH94QxWIa6N4U6MouMmBA" type="text/javascript"></
             script>
12     <script type="text/javascript">
13
14     function initialize() {
15       if (GBrowserIsCompatible()) {
16         var map = new GMap2(document.getElementById("map_canvas"));
17         map.setCenter(new GLatLng(37.4419, -122.1419), 13);
18       }
19     }
20
21     </script>
22   </head>
23   <body onload="initialize()" onunload="GUnload()">
24     <div id="map_canvas" style="width: 500px; height: 300px"></div>
25   </body>
26 </html>
```

Figura 3. Mapa Simples

Vejamos detalhadamente o código da Listagem 1. Nas linhas 01 até 04 estão o cabeçalho de nossa página HTML que possui as características da página. Não entrarei em detalhes sobre esse item por não fazer parte do escopo do artigo. O que nos importa mesmo, está entre as linhas 08 e 21. Nessas linhas há um script JavaScript que carregará a o mapa. Perceba que na verdade passamos alguns parâmetros para a url http://maps.google.com/maps. O único que nos interessa é o parâmetro Key. Ele leva a chave que recebemos ao registrar-se no site do Google Maps, ou seja, nossa licença de uso. Em seguida, linhas 14 a 19, vemos que há outro script JavaScript, dessa vez declarando uma função Initialize(). Ela será chamada mais adiante no evento OnLoad() da página, veja a Linha 23. A função não recebe parâmetros, apenas verifica se o browser é compatível com a API e em seguida cria um mapa usando o método GMap2. A criação é atribuída a uma variável de nome map, que na verdade é um objeto JavaScript. Por fim, o mapa é carregado e centralizado usando as coordenadas 37.4419 e -122.1419. Poderíamos alterar o exemplo do Google recebendo parâmetros na função e passando para ela uma coordenada qualquer. Como podemos ver na Listagem 2. Em seguida bastaria passar as coordenadas no OnLoad() da página, como a seguir:

```
<body onload="initialize(38.2500, -95.100)" onunload="GUnload()">
```

Como podemos ver a utilização não é tão complexa como imaginamos. Para falar a verdade, em meus sistemas acabei utilizando os exemplos da própria documentação do *Google*, fazendo apenas alguns melhoramentos ou mesmo atualizações no layout de visualização. Vamos a um exemplo prático em *Win32*.

Listagem 2. Código simples de página com Google Maps alterado para receber parâmetros

```
01 function initialize(CoordenadaX, CoordenadaY) {
02   if (GBrowserIsCompatible()) {
03     var map = new GMap2(document.getElementById("map_canvas"));
04     map.setCenter(new GLatLng(CoordenadaX, CoordenadaY), 13);
05   }
06 }
```

Mostrando mapas em Win32

Para que possamos usar mapas em aplicações *Win32* teremos que utilizar um macete, já que não conseguimos navegar na Internet com facilidade. Usaremos o componente *WebBrowser* da paleta *Internet* para fazer a navegação. Como precisamos chamar uma página *HTML* para mostrar o mapa, criaremos ela dinamicamente e em seguida a carregaremos para nosso *WebBrowser*, simples assim. Crie uma nova aplicação utilizando o menu *File>New>Application* no *Delphi 7* ou em qualquer outra versão da ferramenta que possuir.

Insira um componente *PageControl* da paleta *Win32*. Não é necessário nomeá-la, apenas adicione ao projeto e inclua duas abas usando o botão direito do mouse e a opção *New Page*. Clique na primeira aba e altere seu *Caption* para *Mapa Simples* e a segunda para *Apontador de Rotas*. Na primeira aba adicione dois *Label's* e dois *Edits* (*edtX*, *edtY*). Limpe a propriedade *Text* dos *Edit's* ou se preferir preencha com os valores *37.4419* e *-122.1419* que na verdade são duas coordenadas válidas já encontradas no exemplo do *Google*. Os dois *Label's* precisam ser alterados somente na propriedade *Caption* conforme podemos ver na **Figura 4.** Insira também um *Button* com o *Caption* igual a *Mostrar*. O *PageControl* precisa ficar alinhado ao topo da tela usando a propriedade *Align*. Por fim, insira um *TWebBrowser* da paleta *Internet* e mude sua propriedade *Align* para *alClient*.

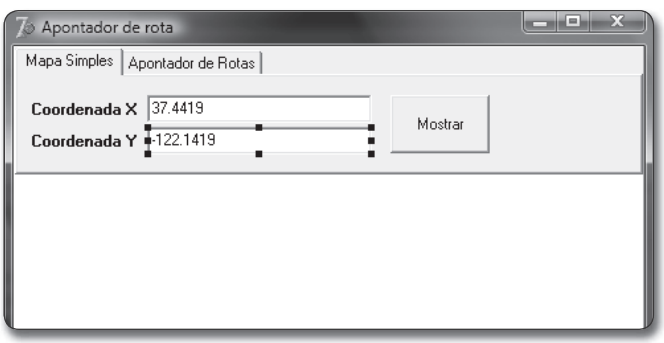


Figura 4. Exemplo de layout para mapa simples

O que faremos agora é copiar o código *HTML* do exemplo *Mapa Simples* para a *Unit* do formulário no *Delphi*. Nós criaremos uma *procedure* chamada *MapaSimples* que fará a criação da página *HTML*, aplicará os parâmetros necessários e então salvará o arquivo *HTML* no diretório da aplicação.

Acesse a sessão *public* do formulário e declare a seguinte *procedure*:

```
procedure MapaSimples(ACoordenadaX, ACoordenadaY: string);
```

Observe que receberemos duas coordenadas, por isso que criamos dois *Edit's* em tela. O botão *Mostrar* enviará para o procedimento as duas coordenadas digitadas nos *edit's*. Faremos então uma concatenação com os valores recebidos e carregaremos a página no *WebBrowser*.

Muito bom, pressione *Ctrl + Shift + C* para que o *Delphi* crie o cabeçalho da nossa *procedure* e vamos ao código. Minha sugestão é que você copie o código *HTML* gerado pela página exemplo do *Google* em nossa *procedure*. Dessa forma, faremos pouca digitação. O que faremos é criar uma variável do tipo *TStringList* e jogar nela todo o código *HTML* já com as alterações necessárias. Salvamos com um nome qualquer e então carregamos no *WebBrowser*. Vamos ao código da **Listagem 3**.

Listagem 3. Código da procedure para montagem do mapa simples

```
01 procedure TForm1.MapaSimples(ACoordenadaX, ACoordenadaY: string);
02 var
03   HTML: TStrings;
04 begin
05   HTML := TStringList.Create;
06   { Cabeçalho do arquivo HTML }
07   with HTML do
08   begin
09     Add(' <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
10       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">');
11     Add(' <html xmlns="http://www.w3.org/1999/xhtml"
12       xmlns:v="urn:schemas-microsoft-com:vml">');
13     Add('   <head>');
14     Add('     <meta http-equiv="content-type" content="text/html;
15          charset=utf-8"/>');
16     Add('     <title>Google Maps JavaScript API Example:
        Simple Map</title>');
17     Add('     <script src="http://maps.google.com/
        maps?file=api&amp;v=2&amp;
18       sensor=false&amp;
19      key=ABQIAAAAzr2EBOXUKnm_jVnkOOJI7xSosDVG8KKPE1-m51RBrvYughuyMxQ-
20         i1QfUnH94QxWIa6N4U6MouMmBA"');
21     Add('                 type="text/javascript"></script>');
22     Add('     <script type="text/javascript">');
23     Add('     function initialize(CoordenadaX, CoordenadaY) {');
24     Add('       if (GBrowserIsCompatible()) {');
25     Add('         var map = new GMap2(document.
        getElementById("map_canvas"));');
26     Add('         map.setCenter(new GLatLng(CoordenadaX,
        CoordenadaY), 13);');
27     Add('       }');
28     Add('     }');
29     Add('     </script>');
30     Add('   </head>');
31     Add('   <body onload="initialize('+ ACoordenadaX + ',' +
32       ACoordenadaY +')" onunload="GUnload()">');
33     Add('     <div id="map_canvas" style="width: 500px;
        height: 300px"></div>');
34     Add('   </body>');
35     Add(' </html>');
36     if FileExists(ExtractFilePath(Application.
        ExeName)+'mapasimples.html') then
37       DeleteFile(ExtractFilePath(Application.ExeName)+'
          mapasimples.html');
38     SaveToFile(ExtractFilePath(Application.ExeName)+'mapasimples.html');
39   end;
40 end;
```

O código é um pouco extenso, mas é bem fácil de entender. Comecemos então pela *linha 03*. Nela declaramos uma variável do tipo *TStrings*. Na *linha 05* estamos instanciando a variável criando uma *TStringList*, ou seja, uma lista de *strings*. Nessa variável vamos jogar todo o código da página que formará nosso arquivo *HTML*. Por isso usamos um bloco *with..do*. Perceba que cada linha do nosso *HTML* é adicionada a variável. O código que estamos vendo é uma cópia do exemplo mapa simples que vimos no tópico anterior. Porém, há uma diferença: na *linha 23* a declaração da função *initialize()* foi modificada para receber dois parâmetros e na *linha 26* esses parâmetros estão sendo utilizados.Outra novidade encontra-se nas *linhas 31* e *32* onde estamos invocando a função *initialize()* no evento *OnLoad()* da página. O restante é código Delphi. Das *linhas 36* até *38*, verificamos se o arquivo *mapasimples.html* existe no diretório da aplicação e o excluímos em caso positivo. Por fim salvamos o novo arquivo. Isso mesmo! Estamos criando um arquivo *HTML* puro via Delphi. Agora a cartada final, no código do evento *OnClick* do botão *Mostrar* digite o código a seguir.

```
MapaSimples(edtX.text, edtY.Text);
WebBrowser1.Navigate(ExtractFilePath(Application.
 ExeName)+'mapasimples.html');
```

Analise com calma e verá que não é muito difícil entender o que fizemos. Chamamos a *procedure MapaSimples* passando como parâmetro o conteúdo dos dois *Edit's*. Por fim, carregamos a página criada usando o método *Navigate* do *WebBroswer*. O resultado pode ser visto na **Figura 5**. Caso tenha instalado o *Google Earth* no seu computador, você pode fazer buscas no mapa e descobrir coordenadas válidas, bastando copiá-las para o programa e mandar *mostrar*.

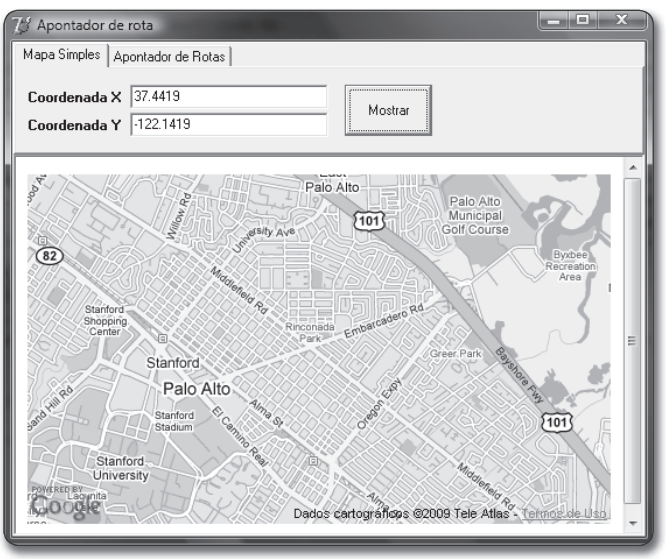


Figura 5. Mapa simples sendo visualizado em aplicação Win32

Criando um apontador de rota

Como podemos ver não há nenhuma dificuldade realmente grande na utilização de mapas com Delphi, portanto vamos avançar um pouco mais em nosso estudo e fazer um mapa um pouco mais complexo. Quando acessamos o link *http://*

Listagem 4. Código para criação do apontador de rotas

```
01 procedure TForm1.CriarMapa(AOrigem, ADestino, ALocalidade: string);
02 var
03   HTML: TStrings;
04 begin
05   HTML := TStringList.Create;
06   { Cabeçalho do arquivo HTML }
07   with HTML do
08   begin
09     ...
10     Add('    <script src=" http://maps.google.com/?
        file=api&amp;v=2.x&amp;
11       key=' + gKey + ''' + ' type="text/javascript"></script>');
12     ...
13     Add('    <script type="text/javascript">');
14     Add('      var map;');
15     Add('      var gdir;');
16     Add('      var geocoder = null;');
17     Add('      var addressMarker;');
18     Add('      function initialize() {');
19     Add('        if (GBrowserIsCompatible()) {');
20     Add('          map = new GMap2(document.getElementById
                      ("map_canvas"));');
21     Add('          gdir = new GDirections(map,
23                      document.getElementById("directions"));');
24     Add('          GEvent.addListener(gdir, "addoverlay",
                      onGDirectionsLoad);');
25     Add('         GEvent.addListener(gdir, "error", handleErrors);');
26     Add('          setDirections(' + ''' + AOrigem + ''' +
        ',' + ''' +
27       ADestino + ''' + ',' + ''' + ALocalidade + ''' + ');');
28     Add('        }');
29     Add('      }');
30     Add('      function setDirections(fromAddress, toAddress,
        locale) {');
31     Add('        gdir.load("from: " + fromAddress + " to: " +
        toAddress,');
32     Add('                    { "locale": locale });');
33     Add('      }');
34     Add('      function onGDirectionsLoad(){');
35     Add('        var poly = gdir.getPolyline();');
36     Add('        if (poly.getVertexCount() > 100) {');
37     Add('          alert("Esta rota possui muitos vértices.");');
38     Add('          return;');
39     Add('        }');
40     Add('        var baseUrl = "http://maps.google.com/
        staticmap?";');
41     Add('        var params = [];');
42     Add('        params.push("center=" +
43       map.getCenter().lat().toFixed(6) + "," +
44       map.getCenter().lng().toFixed(6));');
45     Add('        var markersArray = [];');
46     Add('       markersArray.push(poly.getVertex(0).toUrlValue(5)
       +  ",greena");');
47     Add('        markersArray.push(poly.getVertex(
48       poly.getVertexCount()-1).toUrlValue(5) + ",greenb");');
49     Add('       params.push("markers=" + markersArray.join("|"));');
50     Add('        var polyParams ="rgba:0x0000FF80,weight:5|";');
51     Add('        var polyLatLngs = [];');
52     Add('       for (var j = 0; j < poly.getVertexCount(); j++) {');
53     Add('        polyLatLngs.push(
54       poly.getVertex(j).lat().toFixed(5) +
55       "," + poly.getVertex(j).lng().toFixed(5));');
56     Add('      }');
57     Add('     params.push("path=" + polyParams + polyLatLngs.
        join("|"));');
59     Add('     params.push("zoom=" + map.getZoom());');
60     Add('     params.push("size=310x400");');
61     Add('     var img = document.createElement("img");');
63     Add('     img.src = baseUrl + params.join("&") + "&key=" +
        gKey + "";');
64     Add('    document.getElementById("staticMapIMG").innerHTML =
        "";');
65     Add('     document.getElementById("staticMapIMG").
        appendChild(img);');
66     Add('     document.getElementById("staticMapURL").innerHTML =
67       baseUrl + params.join("&") + "&key=YOUR_KEY_HERE";');
68     Add('  }');
69     Add('  </script>');
70     Add('  </head>');
71     Add('  <body onload="initialize()" onunload="GUnload()">');
72     Add('  <h2>Exemplo Mapas ClubeDelphi</h2>');
73     Add('  <form action="#"
74       onsubmit="setDirections(' + ''' + AOrigem + ''' + ',' +
          ''' + ADestino
75       + ''' + ',' + ''' + ALocalidade + ''' + '); return
          false">');
76     Add('   <input type="hidden" size="25" id="fromAddress"
77       name="from" value="' + ''' + AOrigem + ''' + '"');
78     Add('   <input type="hidden" size="25" id="toAddress"
79       name="to" value="' + ''' + ADestino + ''' + '"');
80     Add('  </form>');
81     Add('    <br/>');
82     Add('    <table class="directions">');
83     Add('    <tr><th>Direcoes</th><th>Mapa</th></tr>');
84     Add('    <tr>');
85     Add('    <td valign="top"><div id="directions" style="width:
86       275px"></div></td>');
87     Add('    <td valign="top"><div id="map_canvas" style="width:
88       500px; height: 450px"></div></td>');
89     Add('    </tr>');
90     Add('    </table>');
91     Add(' <div id="staticMapIMG" style="width: 500px;
92       margin-left: 20px; height: 410px"></div>');
93     Add('<br clear="all"/>');
94     Add('<h3><img src="http://www.google.com/uds/solutions/
          wizards/img/3b.gif"');
95     Add('style="vertical-align:middle"/>Generate URL</h3>');
96     Add('<p>The Static Maps API requires a free API key thats
          associated with
97       your');
98     Add('<a href="http://www.google.com/accounts/">
99       Google Account</a> and the URL of your web site.');
100   Add('You can sign up for one <a');
101    Add('href="http://code.google.com/apis/maps/signup.html">
102      here</a>, and substitute it');
103    Add('as the value of the "&key=" parameter in the URL below.');
104    Add('<br/>');
105    Add('<p>');
106    Add('</p>');
107    Add('<pre id="staticMapURL" class="code">');
108    Add('</pre>');
109    Add('  </body>');
110    Add('</html>');
111
112    if FileExists(ExtractFilePath(Application.ExeName)+'mapa2.
         html') then
113    DeleteFile(ExtractFilePath(Application.ExeName)+'mapa2.
         html');
114    SaveToFile(ExtractFilePath(Application.ExeName)+'mapa2.
         html');
115  end;
116 end;
```

maps.google.com é possível inserirmos a origem e o destino obtendo uma rota. Faremos isso utilizando Delphi, o que é bem interessante, pois podemos traçar rotas usando como origem e destino endereços gravados no banco de dados de nossa aplicação. O exemplo que utilizaremos pode ser visto no link *http://code.google.com/intl/pt-BR/apis/maps/documentation/examples/directions-advanced.html*, portanto, baixe o código *HTML* dessa página ou se preferir digite o código da próxima listagem que veremos.

Selecione a segunda aba da nossa *PageControl* e adicione dois *Edit's* (*edtOrigem*, *edtDestino*) juntamente com dos *Label's* com os *Caption's* iguais a *Origem* e *Destino*, respectivamente. É necessário também um *Button* para acionar a criação da página. Se preferir observe a **Figura 6**.

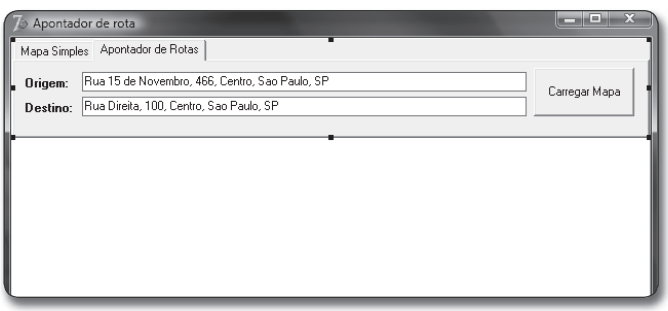


Figura 6. Exemplo de tela para busca de rota

Vamos ao código. Para iniciar declare o procedimento *CriarMapa* como a seguir:

```
procedure CriarMapa(AOrigem, ADestino, ALocalidade: string);
```

Veja que estamos recebendo a *Origem*, o *Destino* e a *Localidade*, que na verdade é a linguagem que o *Google* vai usar para dar o retorno. Nesse caso usaremos *pt-BR* para que tenhamos a página em português. Pressione *Ctrl + Shift + C* para que o Delphi crie o cabeçalho da função então digite o código da **Listagem 4.** Ele é maior em extensão, mas a funcionalidade é bem semelhante.

Eu omiti algumas partes do código-fonte para que possamos entender mais facilmente. No site da *DevMedia* você poder baixar e analisar o código completo da aplicação. Bem, vamos lá. Novamente declaramos uma variável do tipo *TStrings* e a criamos posteriormente. A primeira novidade que encontramos no código é que estamos usando uma constante que eu não mencionei no começo de tópico. Veja as *linhas 10* e *11*. No parâmetro *Key* concatenei com o conteúdo da constante *gKey*. Para isso basta declarar uma constante com esse nome na área *Implementation* da *Unit*, veja:

```
const
  gKey = 'ABQIAAAAjUOEJWnWPMv7oQ-jjS7dYxSPW5CJgpdgO_s4yyMovOaVh_
    KvvhSfpvagV18eOyDWu7VytS6Bi1CWxwc';
```

A constante nada mais é do que a string de licença que recebemos ao nos registrar no site do *Google*. Dessa forma fica mais fácil passar a chave diversas vezes no mesmo sistema. Essa chave poderia estar armazenada em algum campo do banco de dados também, sem problema algum.

Em seguida vemos a declaração da *procedure initialize()*. Dessa vez ela ganhou uma nova roupagem. Agora ela utiliza a *API GDirections* (*Linha 21*) para gerar o mapa. Usamos também o método *setDirections*, onde passamos a *Origem*, *Destino* e a *Localidade*, veja as *linhas 26* e *27*. Entre as *linhas 38* e *69* temos declaradas as funções necessárias para montar o mapa. Não entrarei em detalhes sobre essas funções. Todas elas são usadas internamente pelo *Google Maps*. Na seqüencia encontramos o a chamada ao procedimento *initialize()* no evento *OnLoad()* da página. Veja que concatenamos as *strings* do código com os valores recebidos na *procedure* que são *AOrigem*, *ADestino* e *ALocalidade*. O restante do código até a *linha 111* é apenas código *HTML* para formatar a página incluindo dois boxes que receberão o mapa e a lista de ruas que são necessárias para chegar ao destino.

Entre as *linhas 112* e *114*, fazemos a gravação do arquivo *mapa2.html* no diretório da aplicação. Veremos a seguir que esse arquivo é carregado no *WebBrowser*.

Agora falta fazermos a chamada ao arquivo. No evento *OnClick()* do botão *Mostrar* digite o código a seguir:

```
  CriarMapa(edtOrigem.text, edtDestino.Text, 'pt-BR');
  WebBrowser1.Navigate(ExtractFilePath(Application.ExeName)+'mapa2.
html');
```

Como vemos, é simples. Chamamos a função que criamos anteriormente, *CriarMapa*, e passamos para ela o conteúdo dos dois *Edit's* que possuem, por sua vez, o endereço de origem e destino. Por fim carregamos o arquivo no *WebBrowser*. Os resultados podem ser vistos nas **Figuras 7** e **8.**

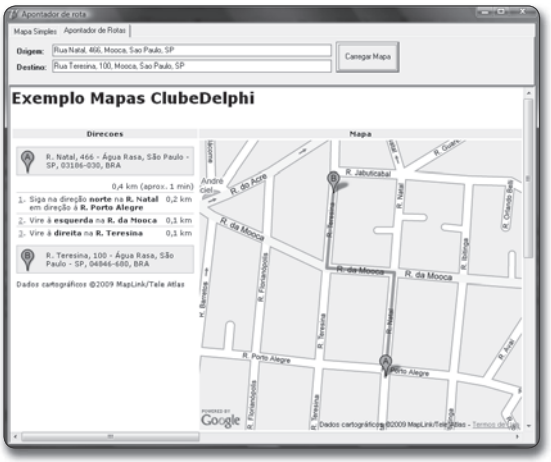


Figura 7. Exemplo de mapa

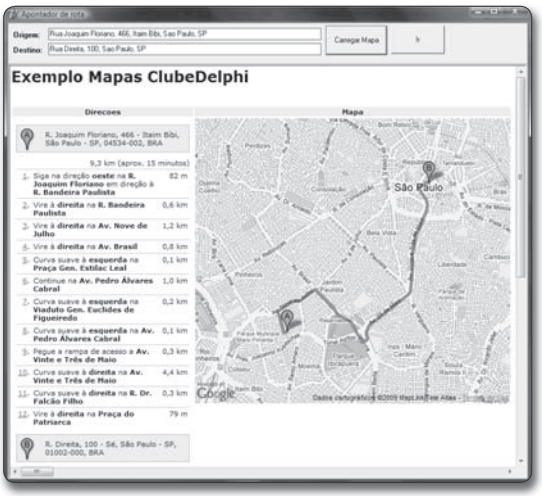


Figura 8. Exemplo de mapa

Como mencionado no início do artigo, há uma grande enciclopédia no site da documentação do *Google Maps*, inclusive em português. Aqui poderíamos muito bem ter substituído os *Edit's* em ambos exemplos por *DBEdit's* ou qualquer outro componente conectado diretamente ao banco de dados. Enfim, basta criatividade e estudo para utilizar o máximo dos recursos de mapas em nossas aplicações.

Conclusão

Nesse artigo vimos como manipular mapas em aplicações *Win32*. Ficou claro que podemos há uma gama enorme de possibilidades para incrementar ainda mais nossas aplicações. Além dos recursos que vimos, ainda é possível, em algumas cidades, navegar pelas ruas em *3D* utilizando o *Google Maps*.

Até mais e boa leitura. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos sobre técnicas que poderão aumentar a qualidade do desenvolvimento de software

Usando dados do BD para gerar Gráficos

Veja como gerar gráficos usando informações do seu banco de dados

Resumo do DevMan

Podemos dizer que todo sistema de informação é baseado em informações advindas do banco de dados da empresa. Isso é fato. Sabemos também, que boa parte de um software é constituída de relatórios que por sua vez enriquecem a aplicação e fazem com que ela tenha um sentido dentro da corporação.

Além os relatórios, muitas empresas se baseiam em gráficos comparativos para demonstrar diversos setores da empresa, tais como vendas, compras, vendas em potencial, vendas antigas, estimativas etc. Por isso nesse artigo veremos como desenvolver facilmente gráficos utilizando o componente TChart que acompanha o Delphi desde os primórdios.

Nesse artigo veremos

- Configuração de algumas das principais propriedades do componente TChart;
- Rotinas para obter os dados de um banco e gerar os gráficos;
- Impressão de mais de um gráfico por página sem o uso de um gerador de relatórios;

Qual a finalidade

Gerar gráficos a partir de informações obtidas em um banco de dados;

Quais situações utilizam esse recurso?

Em aplicações comerciais é comum a necessidade da apresentação de resultados de uma forma mais amigável e de fácil compreensão. Para isso, em geral, utilizamos os gráficos para a exibição de informações obtidas através do cruzamento de dados armazenados no banco;

Ricardo Scache Belardinuci
ri-taqua@ig.com.br
Trabalha há 3 anos como Analista/Programador free-lance na cidade de São Paulo, desenvolvendo os mais variados aplicativos em Delphi para pequenas e médias empresas. Formado em Tecnologia em Processamento de Dados pela FATEC/Taquaritina/SP.

Em quase todos os projetos que trabalhei tive que implementar algum tipo de gráfico além da informação no formato de texto (os tradicionais relatórios). Porém, lembro que na primeira vez que um cliente me pediu para gerar um gráfico eu *gelei*, pois não tinha a menor idéia de como atender aquele pedido. Parti em busca de informações na internet, mas encontrei muito pouco sobre o assunto e a solução foi colocar um *TChart* em um *form*, estudar as suas propriedades e fazer inúmeros testes para entender

o funcionamento do componente. Afim de que outros programadores não tenham as mesmas dificuldades que encontrei, resolvi escrever o que aprendi sobre esse componente. No decorrer do artigo construiremos um pequeno exemplo que preparei, onde veremos algumas das principais propriedades do *TChart*, como gerar um gráfico a partir de informações obtidas em uma base de dados e, por último, como imprimir um ou mais gráficos em uma mesma página sem a utilização de um gerador de relatórios como o *QuickReport*, por exemplo. O aplicativo terá dois gráficos, o primeiro com uma série onde representaremos as vendas de cada mês e um segundo gráfico onde estarão representadas as vendas de cada mês por *Forma de Pagamento* que irá conter três séries.

Neste exemplo utilizaremos o Delphi 7 e será necessário ainda a criação de uma pequena base dados com apenas quatro tabelas. Para a criação do banco utilizei o *Firebird*, porém nada impede que um outro SGDB seja usado.

Configurando o componente TChart

A maior dificuldade com que me deparei ao utilizar o componente *TChart* pela primeira vez foi a grande quantidade de propriedades que o componente possui para serem configuradas. Porém, antes de configurar qualquer propriedade (e isso vale para qualquer componente), é preciso entender o que representa a propriedade. Assim, para facilitar o entendimento dessas propriedades, vamos colocar a mão na massa e montar um pequeno aplicativo. Abra o Delphi 7 e inicie uma nova aplicação. Salve todos os arquivos em uma pasta nomeando a *Unit* do *form* como *ufrmChart.pas* e o projeto como *prjChart.dpr.* Atribua à propriedade *Name* do *form* criado *frmChart* e *Chart* à sua propriedade *Caption.* Coloque nele um componente *TChart*(*chtVendas*) da paleta *Additional*. De um duplo clique sobre o componente para carregar o editor de propriedades do *TChart*. É através desse editor que poderemos criar as séries do gráfico e também configurar suas propriedades.

Para adicionar uma série ao gráfico clique no botão *Add* localizado à direita na página *Chart>Séries.* Escolha a opção *Bar* e clique em *OK*. Ao voltar para o editor teremos a nossa série criada. Para entender melhor o que é uma série, podemos imaginar que ela serve para representar uma determinada informação, assim, se imaginarmos um sistema comercial em que desejamos criar um gráfico que exiba a soma das vendas de cada mês, necessitamos de apenas uma série, pois temos uma única informação, que é o valor das vendas do mês. Por outro lado, se precisamos de um gráfico que apresente as vendas de cada mês separadas por *Forma de Pagamento*, então será necessário uma série para cada *Forma de Pagamento* possível. Portanto, se tivermos uma situação em que as opções de formas de pagamento forem, por exemplo, *CARTÃO, CHEQUE* e *DINHEIRO*, teremos que criar três séries.

Nota: Não confunda a quantidade de barras de um gráfico (considerando um gráfico de barras) com a quantidade de séries a serem criadas. Por exemplo, em um gráfico de vendas por mês, o fato de termos uma barra para cada mês, não quer dizer que para cada um dos meses temos que criar uma nova série. Uma mesma série pode receber diversos valores sendo que para cada valor será gerada uma nova barra.

Depois de criada a série, clique no botão *Title* e nomeie-a para *Vendas*. Selecione a página *Series* do editor e na página *Format* marque a opção *Color Each*. Marcando esta opção teremos uma cor diferenciada para cada barra da nossa série *Venda*. Podemos definir ainda na página *Format* a largura de cada barra atribuindo um valor a propriedade *% Bar Width*, porém vamos deixar o valor *default* para esta e as propriedades restantes desta página.

Na página *General* podemos definir se a legenda será ou não através da propriedade *Show In Legend*, enquanto as opções dos grupos *Horizontal Axis* e *Vertical Axis* determinam o posicionamento dos valores das linhas *X* e *Y*, respectivamente. Nesta página deixe os valores padrão. Na página *Marks* (**Figura 1**) temos várias propriedades que irão interferir no aspecto visual do nosso gráfico e também que informações serão exibidas.

Começamos pela opção *Visible*, que define se os valores serão ou não apresentados logo acima de cada barra. Deixamos esta opção marcada, mas para efeito de aprendizado experimente desmarcar/marcar esta opção e veja o efeito. Dentro do grupo *Format*, temos a opção *Transparent* que se marcada tornará transparente o fundo do retângulo que envolve os valores de cada barra. Ao lado desta opção temos o botão *Back Color* que define a cor de fundo dos valores. Para nosso exemplo vamos deixar a opção *Transparent* marcada.

No botão *Font*, podemos definir a fonte e a cor dos valores. Clicando no botão *Border* irá aparecer uma pequena janela (**Figura 2**) onde poderemos definir a cor, estilo, largura e visibilidade da borda que envolve os valores. Vamos retirá-la deixando a opção *Visible* desmarcada. Dentro do grupo *Arrows*, temos o botão *Color* que ao ser clicado exibe uma pequena janela onde podemos definir a cor, largura e visibilidade da linha existente entre cada barra do gráfico e o seu respectivo valor logo acima. Para nosso exemplo, retire esta linha desmarcando a o opção *Visible* desta janela. Temos ainda nesse grupo a propriedade *Length* que define a distância entre o valor e a barra. Altere esta propriedade para 10. Por fim, temos no lado esquerdo o grupo *Style*, onde definimos o que será exibido acima de cada barra do gráfico. Dentro deste grupo destacamos as opções *Value*, que exibe os valores absolutos de cada barra, *Percent*, que exibe o percentual de cada barra, *Label*, que irá exibir o rótulo, *Label and Value*, exibe o rótulo e o seu respectivo valor e a opção *Label & Percent Total* que exibirá o rótulo e o percentual do total.

Para o nosso exemplo, vamos marcar a opção *Value*, mas para melhor entender altere as opções e veja as mudanças que ocorrem no gráfico.

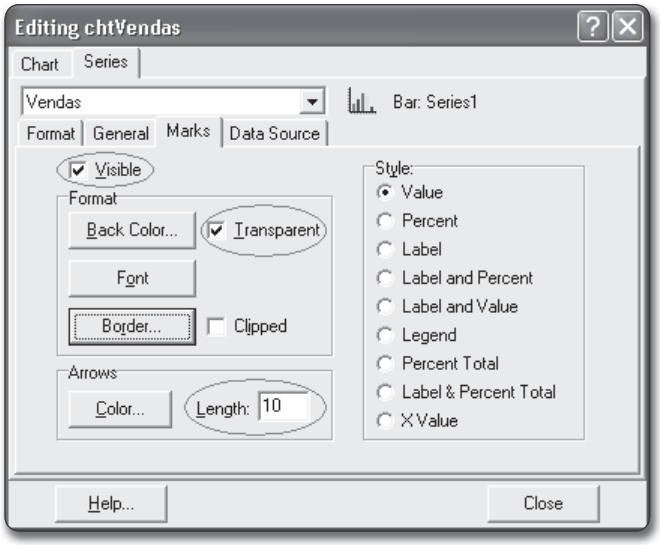


Figura 1. Definindo as propriedades da página Marks

Figura 2. Editando a borda dos valores

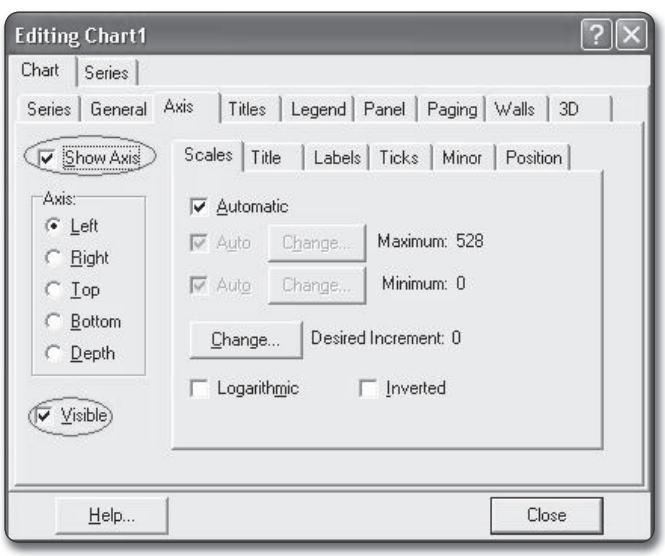


Figura 3. Configurando as opções da página Axis

Dando seqüência ao nosso exemplo, selecione a página *Chart* e em seguida clique em *Axis* (**Figura 3**). Nesta página encontramos uma quantidade enorme de opções de configuração do gráfico. Começamos pela opção *Show Axis* na parte superior à esquerda, que define a visibilidade dos valores dos eixos x e y assim como todas as linhas pontilhadas na horizontal e vertical do gráfico. Desmarcando esta opção teremos um gráfico mais *limpo*. Para nosso exemplo, deixe esta opção marcada.

Logo abaixo da opção Show Axis, temos o grupo Axis que permite configurar cada um dos lados do gráfico separadamente. Por exemplo, se a opção Left estiver selecionada e desmarcarmos a opção Visible (localizada abaixo), os valores a esquerda do gráfico (eixo y) são ocultadas, enquanto que se a opção Bottom estiver selecionada, os valores da linha inferior (eixo x) é que serão ocultadas. Vamos deixar a opção Visible marcada. Dentro da página Axis temos ao centro um outro conjunto de páginas. Selecione a opção Title deste conjunto, e com a opção Left do grupo Axis à esquerda marcada, digite Valores na caixa de texto ao lado de Title.

Note que irá aparecer à esquerda do gráfico, na vertical, o texto digitado. Agora selecione a opção *Bottom* do grupo *Axis* e digite *Meses* na mesma caixa de texto. Teremos o texto na parte inferior do gráfico, logo abaixo dos valores do eixo x. Por questões de espaço vamos deixar a propriedade *Titles* em branco. Na página *Labels* podemos regular o ângulo dos valores dos eixos x e y e também alterar suas propriedades de *font*. Selecione a opção *Bottom* (do grupo *Axis*) e altere a propriedade *angle* para 325, assim teremos os valores na parte inferior (eixo x) em uma posição de 325º. Clique no botão *Font* e altere a cor da fonte para Azul. Agora vá a página *Ticks* e com a opção *left* do grupo *Axis* selecionada, clique no botão *Grid Border*. Abrirá uma janela onde podemos definir a cor, espessura, estilo e visibilidade das linhas pontilhadas horizontais que cortam o gráfico.

Para termos um gráfico mais *limpo* vamos desmarcar a opção *Visible* desta janela. Selecione a opção *Bottom* no grupo *Axis* e clique novamente no botão *Grid Border* e também desmarque a opção *Visible* para ocultarmos as linhas pontilhadas verticais do gráfico. Ainda na página *Ticks* podemos ocultar, definir a cor e largura das bordas do gráfico, selecionando uma das opções do grupo *Axis* e clicando no botão *Axis Border* para configurar as propriedades.

Para atribuir um título ao gráfico clique na página *Titles* (ao lado da página *Axis*). Nesta página podemos, além de atribuir o título, configurar a sua fonte, cor, posicionamento, borda e sua visibilidade. O título do gráfico deve ser digitado na caixa de texto na parte inferior da página. Destacamos nesta página ainda os botões *Font* e *Border* que permitem configurar a fonte e a borda do título, respectivamente e o grupo *Alignment* que define o posicionamento do título. Temos na página *Titles* ainda uma caixa de seleção na parte superior onde podermos alternar entre os valores *Title* e *Foot*. Selecionando a opção *Foot* podemos configurar

um texto (da mesma forma que configuramos o título do gráfico) que será exibido no rodapé. Para o nosso primeiro gráfico altere o título para *Gráfico de Vendas* com a fonte na cor preto e em negrito. As outras opções deixe com o valor default.

Configurando a legenda

A legenda é uma das principais partes do gráfico e mostra informações sobre as séries. Sem a legenda é praticamente impossível entender perfeitamente o que o gráfico quer dizer. Vamos agora ver e alterar algumas das propriedades relacionadas a legenda do gráfico. Para isso, selecione a página *Legend*. Nessa página temos logo acima, no canto superior esquerdo, a opção *Visible*, que define a visibilidade da legenda. Para este primeiro gráfico, deixe esta opção desmarcada. Temos também, logo abaixo da opção *Visible*, 3 botões: *Back Color*, que ao clicarmos abre uma janela para configurarmos a cor de fundo da legenda, *Font*, que permite alterar as propriedades de fonte da legenda e o botão *Frame* que define a cor, estilo e visibilidade da linha que contorna a legenda. Deixaremos as opções *default* para estas propriedades.

No grupo *Position* (abaixo do botão *Frame*), podemos definir o posicionamento da legenda: *Left* (esquerda), *Right* (direita) *Top* (acima) e *Bottom* (em baixo). Vamos selecionar a opção *Bottom* para essa propriedade. Dentro grupo *Shadow* temos o botão *Color*, onde podemos definir a cor do sombreamento da legenda, enquanto que a propriedade *Size*, dentro desse mesmo grupo, define a largura desse sombreamento. Vamos eliminar essa sombra atribuindo o valor 0 à propriedade *Size*.

Podemos também alterar o tamanho das cores da legenda através da propriedade *% Color Width*. Configure essa propriedade com o valor 20 para termos uma maior visibilidade das cores da legenda. Na página *Panel*, podemos dar um efeito gradiente para o painel de fundo do gráfico. Dentro do grupo *Gradient*, temos a opção *Visible* que ativa/desativa o efeito e habilita os botões *Start Color* e *End Color* que definem as cores iniciais e finais, respectivamente. Para o nosso exemplo ative o efeito gradiente e altere a cor final (Botão *End Color*) para cinza 25%.

Agora selecione a página *Walls* onde podemos configurar o efeito de *muro* ao fundo gráfico. A opção *Visible* desta página define a visibilidade do efeito *muro*. Temos ao centro um conjunto de 3 páginas para configurar cada um dos lados do muro. Dentro de cada uma dessas páginas temos 3 botões:

- *Background*: que define a cor de fundo;
- *Border*: para configurar as bordas;
- *Pattern:* para definir o estilo.

Temos ainda a opção *Transparent* que inutiliza as configurações definidas em cada um dos botões deixando as bordas transparentes. Vamos ocultar o muro ao fundo desmarcando a opção *Visible* desta página. Para as outras opções deixe o valor padrão.

Com podemos notar até aqui, não há grandes dificuldades em se montar gráficos no Delphi. Além do que já vimos, temos a página *3D*. Nesta página podemos retirar o efeito tridimensional das barras desmarcando a opção *3D Dimensions*. Podemos também alterar o percentual do efeito 3D modificando o valor da propriedade *3D %*. Deixe os valores default nessa página. Feita todas as configurações deveremos ter um gráfico semelhante à **Figura 4**.

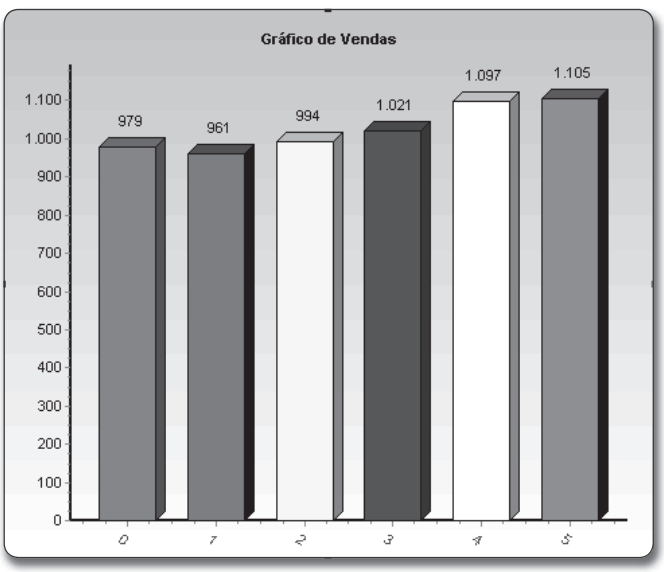


Figura 4. Visão do gráfico após as configurações realizadas

Vamos agora configurar nosso segundo gráfico que será muito semelhante ao primeiro, com algumas diferenças que iremos detalhar a seguir. Primeiro insira um novo componente *TChart* no *form* (*frmChart*) e posicione-o à direita do primeiro. Defina sua altura e largura igual ao primeiro e nomeie-o para *chtVendasFormaPagto*. Para esse segundo gráfico vamos passar as propriedades que devem ser configuradas diferentes do primeiro, portanto, as configurações não mencionadas deverão ser iguais ao gráfico de vendas (*chtVendas*).

A primeira diferença deste segundo gráfico é que teremos 3 séries ao invés de uma como no gráfico de vendas. As séries serão criadas para representar as formas de pagamento possíveis em nosso exemplo: *Cartão*, *Cheque* e *Dinheiro*. Dê um duplo clique sobre o componente para abrir o editor de propriedades. Na pagina *Chart>Series* adicione 3 séries clicando no botão *Add* à direita e selecionando sempre o gráfico do tipo *Bar*.

Após adicionar as séries altere o título de cada uma clicando no botão *Title*. As séries deverão ter seus títulos alterados para *Cartão*, *Cheque* e *Dinheiro*. Quando trabalhamos com gráficos com mais de uma série, algumas propriedades precisam ser configuradas individualmente para cada uma das séries criadas. Vá até a página *Series* (ao lado de Chart) e note que há uma caixa de seleção onde temos todas séries que criamos. Em todas elas a opção *Color Each* deverá ficar

desmarcada, na página *Marks*, retire as linhas que envolvem os valores acima de cada barra da mesma maneira como foi feito em nosso primeiro gráfico (*chtVendas*) e selecione a opção *Value* no grupo *Style*.

As alterações a seguir não precisam ser realizadas para cada série. Na página *Legend*, marque a opção *Visible* para exibir a legenda e configure o restante das propriedades desta página igual ao gráfico de vendas (*chtVendas*). Por fim, altere o titulo do gráfico (na página *Titles*) para *Gráfico de Vendas Por Forma de Pagamento*. Como foi dito anteriormente, as outras configurações deveram ser feitas igual ao nosso primeiro gráfico.

Agora que já configuramos os dois gráficos do nosso exemplo, adicione um *Panel* (paleta *Standard*) ao nosso *form*, configure sua propriedade *Align* para *alTop* e deixe sua propriedade *Caption* em branco. Selecione o gráfico de vendas (*chtVendas*) e altere sua propriedade *Align* para *alLeft*. Para o segundo gráfico (*chtVendasFormaPagto*) altere a propriedade *Align* para *alRight*. Adicione um componente *TGroupBox* (paleta *Standard*) e altere seu *Caption* para *Período de Pesquisa*.

Dentro do *Groupbox* coloque dois componentes *TCombobox*, renomeio-os para *cmbMesIni* e *cmbMesFim*, adicione à propriedade *Items* todos os meses do ano, modifique a propriedade *Style* para *csDropDownList* e altere o *ItemIndex* para *0*. Inclua dois componentes *TEdit* (paleta *Standard*) nomeando-os para *edtAnoIni* e *edtAnoFim*. Coloque 2 componentes *TLabel* e posicione-os logo acima de cada um dos componentes *TCombobox*. Altere os *Captions* dos *Labels* para *Mês/Ano (Início)* e *Mês/Ano (Fim)*. Adicione ainda um *TButton* (*btnGerarGrafico*) da paleta *Standard* e altere seu *Caption* para *Gerar Gráficos*.

Concluída esta parte de configuração dos gráficos e montagem da tela, nosso *form* deverá ter uma aparência semelhante à **Figura 5**.

Figura 5. Aparência da aplicação após as configurações.

Gerando gráficos com base em BD

Até o momento vimos como criar gráficos manualmente e com valores pré-definidos. Como o objetivo do nosso exemplo é gerar um gráfico a partir de informações obtidas em uma base de dados, criamos uma pequena estrutura em *Firebird* (*Basedados.fdb*). Não entrarei em detalhes quanto a criação do banco por não fazer parte do escopo desse artigo. Crie um banco de dados seguindo as tabelas descritas na **Figura 6,** ou se preferir baixe o BD direto do site da DevMedia no link dessa edição.

Nota: A base de dados poderá ser criada em qualquer SGBD. Para o nosso exemplo é necessário apenas seguir a estrutura das tabelas.

Adicione ao *form* um componente *TSQLConnection* (*sqlConexao*) da paleta *dbExpress* e configure uma conexão com a nossa base de dados. Na seqüência inclua dois componentes *SQLQuery* nomeando-os para *qryVendas* e *qryVendasFormaPagto*, dois componentes *DataSetProvider*(*dspVendas* e *dspVendasFormaPagto*) e mais três componentes *ClientDataSet*(*cdsVendas*, *cdsVendasFormaPagto* e *cdsAuxliar*). Configure os componentes da seguinte forma: atribua *sqlConexao* para a propriedade *SQLConnection* da *qryVendas* e *qryVendasFormaPagto*. Para a propriedade *DataSet* do *dspVendas* coloque *qryVendas* e para o *dspVendasFormaPagto* coloque *qryVendasFormaPagto*. Para a propriedade *ProviderName* do *cdsVendas* atribua *dspVendas* e para o *cdsVendasFormaPagto* coloque *dspVendasFormaPagto*. Dê um duplo clique sobre o *cdsAuxiliar* e no editor de campos clique com o botão direito e selecione *New Field* para criar cinco novos campos. Os campos deverão ser: MÊS (*string*), Ano(*string*), CARTAO (*Float*), CHEQUE (*Float*) e DINHEIRO (*Float*).

Feitas as ligações entre os componentes de acesso, vamos atribuir os comandos *SQL* aos componentes *sqlVendas* e *sqlVendasFormaPagto*. Vamos começar pelo *sqlVendas* digitando em sua propriedade a instrução da **Listagem 1**.

Listagem 1. Instrução SQL de sqlVendas

```
SELECT
  SUM(IV.TOTAL_ITEM) SOMA,
  EXTRACT(MONTH FROM V.DATA) MES,
  EXTRACT(YEAR FROM V.DATA) ANO
FROM
  ITENS_VENDA IV
  INNER JOIN VENDAS V ON
  IV.ID_VENDA = V.ID_VENDA
WHERE
  EXTRACT(MONTH FROM V.DATA) >= :MES_INI
  AND EXTRACT(YEAR FROM V.DATA) >= :ANO_INI
  AND EXTRACT(MONTH FROM V.DATA) <= :MES_FIM
  AND EXTRACT(YEAR FROM V.DATA) <= :ANO_FIM
GROUP BY
  EXTRACT(YEAR FROM V.DATA),
  EXTRACT(MONTH FROM V.DATA)
```

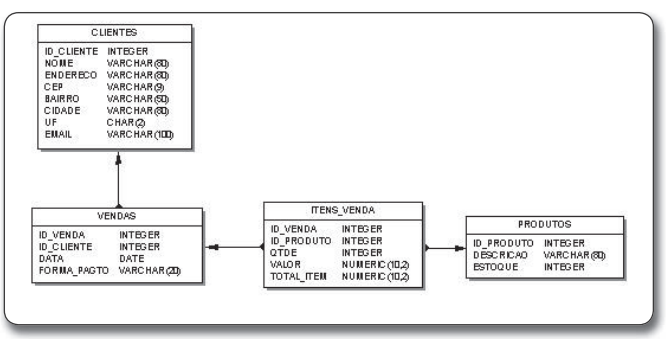


Figura 6. Diagrama das tabelas da base dados

Perceba que estamos selecionando os dados das tabelas *Itens_Venda* e *Vendas* usando um *Join*. Após adicionar o comando *SQL* clique na propriedade *Params* da *qryVendas*, selecione todos os parâmetros que foram criados e configure a propriedade *ParamType* com *ptInput* e *DataType* com *ftInteger*. Em seguida carregue todos os campos dando um duplo clique no componente *qryVendas* e depois selecionando *Add all Fields* ao clicar com o botão direito sobre o editor de campos. Repita os mesmos passos para carregar os campos do *ClientDataSet cdsVendas*. Clique com o botão direito sobre o *cdsVendas* e depois clique em *Fetch Params* para carregar os parâmetros para o *ClientDataSet*.

Nota: No comando *SQL* da *qryVendas* estamos selecionando a soma de todas as vendas e agrupando-as por Ano e mês. Na cláusula *Where* colocamos alguns parâmetros para limitar o período da pesquisa.

Vamos agora atribuir o comando SQL à *qryvendasFormaPagto*. Na propriedade SQL do componente digite a **Listagem 2**.

Listagem 2. Instrução SQL de qryvendasFormaPagto

```
SELECT
  SUM(IV.TOTAL_ITEM) SOMA,
  EXTRACT(MONTH FROM V.DATA) MES,
  EXTRACT(YEAR FROM V.DATA) ANO,
  V.FORMA_PAGTO
FROM
  ITENS_VENDA IV
  INNER JOIN VENDAS V ON
  IV.ID_VENDA = V.ID_VENDA
WHERE
  EXTRACT(MONTH FROM V.DATA) >= :MES_INI
  AND EXTRACT(YEAR FROM V.DATA) >= :ANO_INI
  AND EXTRACT(MONTH FROM V.DATA) <= :MES_FIM
  AND EXTRACT(YEAR FROM V.DATA) <= :ANO_FIM
GROUP BY
  EXTRACT(YEAR FROM V.DATA),
  EXTRACT(MONTH FROM V.DATA),
  V.FORMA_PAGTO
```

Agora perceba que a instrução é semelhante, porém dessa vez estamos usando o campo *Forms_Pagto* para que possamos selecionar os dados agrupando por forma de pagamento. Na seqüência carregue os campos da *qryVendasFormaPagto* e do *cdsVendasFormaPagto* e configure os parâmetros da mesma forma que configuramos a *qryVendas* e *cdsVendas*.

Com nossos componentes de acesso aos dados configurados, vamos implementar as rotinas para a geração dos gráficos. Nosso primeiro passo será criar uma rotina para gerar o gráfico de *Vendas*. Na **Listagem 3** temos o código completo do procedimento.

O código do procedimento é bem simples. Primeiro verificamos se o *cdsVenda* está aberto e, caso esteja, o fechamos. Em seguida atribuímos valores aos parâmetros do *cdsVenda* e então o abrimos chamando o método *Open*. Limpamos as *séries* do gráfico para na seqüência percorrer os registros do *cdsVenda* e passar os valores para a série do *chtVenda* através do método *AddY*. Temos ainda dentro do *laço for* uma chamada à função *NomeMes* que retorna o nome abreviado do

Listagem 3. Código da rotina de geração do gráfico de Vendas

```
procedure TfrmChart.GerarGraficoVendas(pMesIni, pAnoIni,
pMesFim,
  pAnoFim: integer);
var
  Valor: Double;
  Mes,
  Ano: string;
begin
  if cdsVendas.Active then
     cdsVendas.Close;
  cdsVendas.Params.ParamByName('MES_INI').AsInteger:= pMesIni;
  cdsVendas.Params.ParamByName('ANO_INI').AsInteger:= pAnoIni;
  cdsVendas.Params.ParamByName('MES_FIM').AsInteger:= pMesFim;
  cdsVendas.Params.ParamByName('ANO_FIM').AsInteger:= pAnoFim;
  cdsVendas.Open;
  chtVendas.Series[0].Clear;
  while not cdsVendas.Eof do
  begin
    Valor:= cdsVendasSOMA.AsFloat;
    Mes:= NomeMes(cdsVendasMES.AsInteger);
    Ano:= cdsVendasANO.AsString;
    chtVendas.Series[0].AddY(Valor, Mes+'/'+Ano);
    cdsVendas.Next;
  end;
end;
```

Listagem 4. Código da função NomeMes

```
function TfrmChart.NomeMes(pNumMes: integer): string;
var
   aMes: array [1..12] of string;
begin
  aMes[1] := 'Jan';
  aMes[2] := 'Fev';
  aMes[3] := 'Mar';
  aMes[4] := 'Abr';
  aMes[5] := 'Mai';
  aMes[6] := 'Jun';
  aMes[7] := 'Jul';
  aMes[8] := 'Ago';
  aMes[9] := 'Set';
  aMes[10]:= 'Out';
  aMes[11]:= 'Nov';
  aMes[12]:= 'Dez';
  Result:= aMes[pNumMes];
end;
```

mês conforme o número que é passado no parâmetro. Veja na **Listagem 4** o código da função *NomeMes*.

A função *NomeMes* é formada por *array* onde cada posição armazena o nome abreviado do mês correspondente ao número da posição no *array*.

Nota: É perfeitamente possível pegar o nome do mês abreviado usando o método *FormatDateTime*, porém em nosso caso temos apenas o número do mês e a função *FormatDateTime* pede como primeiro parâmetro uma variável que contenha a data completa, ou seja, um *DateTime*. Além disso, a função retorna o nome do mês na linguagem do sistema operacional, resultando em *Jan, Feb..Sep..Dec*, caso o *Windows* esteja inglês.

Para gerar o nosso segundo gráfico, que exibirá a soma das vendas de cada mês divididas por *Forma de Pagamento*, deveremos criar duas novas rotinas: *MontarDataSet* e *GerarGraficoVendasFormaPagto*. Veja nas **Listagens 5 e 6** o código completo dos dois procedimentos.

No código da **Listagem 5** nós criamos um *DataSet* fazendo uma chamada ao método *CreateDataSet* do *cdsAuxiliar* e depois inserimos registros nele com dados obtidos do *cdsVendasFormaPagto*. O objetivo é termos no *cdsAuxiliar* um *DataSet* onde cada registro armazene o mês, ano, soma das vendas à cartão, cheque e dinheiro.

Listagem 5. Código do procedimento MontarDataSet

```
procedure TfrmChart.MontarDataSet;
var
  lFormaPagto: string;
  lMes,
  lANo: integer;
begin
  lMes:= 0;
  lANo:= 0;
  cdsAuxiliar.Close;
  cdsAuxiliar.CreateDataSet;
  cdsVendasFormaPagto.Last;
  while not cdsVendasFormaPagto.Bof do
  begin
    if (lMes <> cdsVendasFormaPagtoMES.AsInteger) or
       (lANo <> cdsVendasFormaPagtoANO.AsInteger) then
    begin
      if (cdsAuxiliar.State = dsInsert) then
        cdsAuxiliar.Post;
        lMes:= cdsVendasFormaPagtoMES.AsInteger;
        lANo:= cdsVendasFormaPagtoANO.AsInteger;
        cdsAuxiliar.Insert;
        cdsAuxiliarMES.AsString:= NomeMes(lMes);
        cdsAuxiliarANO.AsString:= cdsVendasFormaPagtoANO.
         AsString;
     end;
     lFormaPagto:= cdsVendasFormaPagtoFORMA_PAGTO.AsString;
     if (lFormaPagto = 'CARTÃO') then
       cdsAuxiliarCARTAO.AsFloat:= cdsVendasFormaPagtoSOMA.
        AsFloat
     else if (lFormaPagto = 'CHEQUE') then
       cdsAuxiliarCHEQUE.AsFloat:= cdsVendasFormaPagtoSOMA.
        AsFloat
     else if (lFormaPagto = 'DINHEIRO') then
       cdsAuxiliarDINHEIRO.AsFloat:= cdsVendasFormaPagtoSOMA.
        AsFloat;
     cdsVendasFormaPagto.Prior;
  end;
     if (cdsAuxiliar.State = dsInsert) then
      cdsAuxiliar.Post;
end;
```

Listagem 6. Código do procedimento GerarGraficoVendasFormaPagto

```
procedure TfrmChart.GerarGraficoVendasFormaPagto(pMesIni,
pAnoIni, pMesFim,
  pAnoFim: integer);
var
  Mes,
  Ano: string;
begin
  if cdsVendasFormaPagto.Active then
     cdsVendasFormaPagto.Close;
  cdsVendasFormaPagto.Params.ParamByName('MES_INI').AsInteger:=
   pMesIni;
  cdsVendasFormaPagto.Params.ParamByName('ANO_INI').AsInteger:=
   pAnoIni;
  cdsVendasFormaPagto.Params.ParamByName('MES_FIM').AsInteger:=
   pMesFim;
  cdsVendasFormaPagto.Params.ParamByName('ANO_FIM').AsInteger:=
   pAnoFim;
  cdsVendasFormaPagto.Open;
  MontarDataSet;
  chtVendasFormaPagto.Series[0].Clear;
  chtVendasFormaPagto.Series[1].Clear;
  chtVendasFormaPagto.Series[2].Clear;
  while not cdsAuxiliar.Eof do
  begin
    Mes:= cdsAuxiliarMES.AsString;
    Ano:= cdsAuxiliarANO.AsString;
    chtVendasFormaPagto.Series[0].AddY(cdsAuxiliarCARTAO.
     AsFloat, Mes+'/'+Ano);
    chtVendasFormaPagto.Series[1].AddY(cdsAuxiliarCHEQUE.
     AsFloat, Mes+'/'+Ano);
    chtVendasFormaPagto.Series[2].AddY(cdsAuxiliarDINHEIRO.
     AsFloat, Mes+'/'+Ano);
    cdsAuxiliar.Next;
  end;
end;
```

Dentro da rotina é feita uma verificação para que a cada mês e ano seja inserido um novo registro no *DataSet*. Para que os registros no *cdsAuxiliar* fossem gravados em uma ordem cronológica crescente, posicionamos o cursor no último registro do *cdsVendasFormaPagto* e depois percorremos os registros até o primeiro. Entre as linhas de código foram inseridos comentários para facilitar o entendimento do procedimento.

O código da **Listagem 6** é muito semelhante a rotina *GerarGraficoVendas* (**Listagem 3**). A diferença é que temos uma chamada à rotina *MontarDataSet* e ao percorrer os registros do *cdsAuxliar* atribuímos os valores às três séries do *chtVendasFormaPagto*.

Agora que montamos os procedimentos necessários para a geração dos nossos gráficos vamos fazer a chamada dessas rotinas no evento *OnClick* do botão *Gerar Gráficos* (*btnGerarGraficos*). Porém, antes de fazermos as chamadas aos procedimentos, armazenamos em variáveis locais o mês e ano inicial e final conforme os valores dos componentes *cmbMesIni*, *cmbMesFim*, *edtAnoIni* e *edtAnoFim* e depois passamos nos parâmetros do procedimento. Na **Listagem 7** temos o código completo do botão *Gerar Gráfico*.

Listagem 7. Código do evento OnClick do btnGerarGrafico

```
procedure TfrmChart.btnGerarGraficoClick(Sender: TObject);
var
  aMesI,
  aMesF,
  aAnoI,
  aAnoF: integer;
begin
  aMesI:= cmbMesIni.ItemIndex + 1;
  aMesF:= cmbMesFim.ItemIndex + 1;
  aAnoI:= StrToIntDef(edtAnoIni.Text,2009);
  aAnoF:= StrToIntDef(edtAnoFim.Text,2009);
  GerarGraficoVendas(aMesI, aAnoI, aMesF, aAnoF);
  GerarGraficoVendasFormaPagto(aMesI, aAnoI, aMesF, aAnoF);
end;
```

O que fazemos é bem simples, veja que declaramos algumas variáveis para receber o mês e ano inicial e final e logo em seguida chamamos os gráficos usando os métodos que criamos, *GerarGraficoVendas* e *GerarGraficoVendasFormaPagto*. Execute o programa e faça um teste clicando no botão *Gerar Gráficos*. Veja na **Figura 7** o aplicativo em execução.

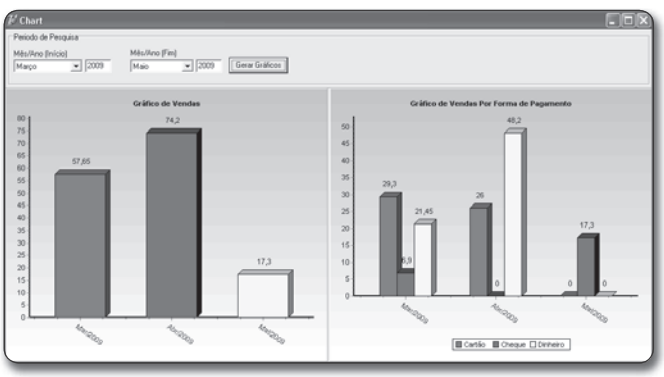


Figura 7. Aplicativo exemplo em execução

Impressão de 2 gráficos na mesma página

Para finalizar o nosso exemplo, vamos construir uma rotina para impressão dos dois gráficos em uma mesma página sem o uso de um gerador de relatórios. Isso mesmo, usaremos os métodos da *Unit Printers* do Delphi. Veja na **Listagem 8** o código.

Como mencionado, na **Listagem 8** fazemos uso do gerenciador de impressão *Printers* para imprimir os gráficos. Declare na

Listagem 8. Rotina para impressão dos gráficos

```
procedure Imprimir(pCht1, pCht2: TChart; pTitulo: string);
var
  tmpH: Longint;
  tmpW: Longint;
  tmpWMargin: Longint;
  tmpHMargin: Longint;
  OldOrientation: TPrinterOrientation;
  Tamanho: Longint;
  Inicio: Longint;
begin
  Screen.Cursor:= crHourGlass;
  OldOrientation:= Printer.Orientation;
  Printer.Orientation:= poLandscape;
  try
    Printer.BeginDoc;
    try
      Printer.Canvas.Font.Size:=11;
      Printer.Canvas.Font.Name:='Arial';
      Printer.Canvas.Font.Color:= clNavy;
      Printer.Canvas.Font.Style:= [fsBold];
      Tamanho:=Printer.PageWidth;
      Tamanho:=Round(Tamanho/2);
      Inicio:= Tamanho - Round(Printer.Canvas.
TextWidth(pTitulo)/2);
      Printer.Canvas.TextOut(Inicio,30,pTitulo);
      pCht1.PrintResolution:= 0;
      pCht2.PrintResolution:= 0;
      tmpW:=Printer.PageWidth;
      tmpWMargin:=Round(5.0*tmpW/100.0);
      tmpW:=tmpW-2*tmpWMargin;
      tmpW:=tmpW div 2;
      tmpH:=Printer.PageHeight;
      tmpHMargin:=Round(5.0*tmpH/100.0);
      tmpH:=tmpH-2*tmpHMargin;
      tmpH:=tmpH div 2;
      pCht1.PrintPartial(Rect(tmpWMargin,tmpHMargin,
        tmpWMargin+tmpW,tmpHMargin+tmpH));
      pCht2.PrintPartial(Rect(tmpWMargin+tmpW,tmpHMargin,
        tmpWMargin+2*tmpW,tmpHMargin+tmpH));
      Printer.EndDoc;
    except on Exception do
      begin
        Printer.Abort;
        Printer.EndDoc;
        raise;
      end;
    end;
  finally
    Printer.Orientation:=OldOrientation;
    Screen.Cursor:=crDefault;
  end;
end;
```

primeira cláusula uses do *form* a *Unit Printer.* Primeiramente definimos a orientação do papel (*Horizontal*), em seguida definimos as propriedades de fonte do título que será impresso no topo da página. Na seqüência é feito o cálculo das margens *Horizontais* e *Verticais* para o posicionamento dos gráficos (um à esquerda e outro à direita na parte superior da página) .

Inclua um novo botão (*btnImprimir*) no *form* e no evento *OnClick* faça a chamada do procedimento *Imprimir* como a seguir. Execute o programa, gere os gráficos e faça um teste de impressão.

```
  Imprimir(chtVendas, chtVendasFormaPagto, 'Exemplo TChart -
DevMedia' );
```

Conclusão

Vimos neste artigo como podemos apresentar de forma amigável e de fácil compreensão informações armazenadas em um banco de dados através da geração de gráficos. Utilizando a mesma base de dados do exemplo, poderíamos ainda gerar outros gráficos como, por exemplo, um que exibisse a soma das vendas por forma de pagamento dentro de um período sem separá-las por mês. Uma alternativa interessante seria exibir um gráfico com os 10 produtos mais vendidos em um dado período. Bom, fica a dica para quem quiser se aventurar na criação de novos gráficos. Abraço a todos e até a próxima. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

Compactando com PHP

Saiba como compactar arquivos usando o PHP

Nesse artigo veremos

- Como utilizar a biblioteca ZLib para compactação;
- Como compactar arquivos usando a biblioteca ZLib.

Qual a finalidade

Entender o funcionamento da biblioteca ZLib que tem como principal objetivo compactar arquivos on-line;

Quais situações utilizam esse recurso?

Podemos aplicar esses recursos em diversas situações. Uma delas pode ser a criação de backups de sites inteiros on-line;

Adriano Santos
falecom@adrianosantos.pro.br
É desenvolvedor Delphi desde 1998, professor e programador PHP, bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É Editor Chefe das revistas ClubeDelphi e WebMobile. Gerente de TI na SoftPark, parceira Borland, membro fundador do DUG-SP (www.dug-sp.com) e mantenedor do blog Delphi to Delphi (www.delphitodelphi.blogspot.com) com dicas, informações e tudo sobre o desenvolvimento Delphi.

No *Windows*, ou mesmo no *Linux*, é comum utilizar-se de programas para compactação de arquivos, tais como *Winzip*, *7Zip*, *WinRar* ou *Brazip*, enfim. O hábito de compactar arquivos é antigo e desde os primórdios da computação utilizamos desse recurso para efetuar backups ou simplesmente para enviar arquivos mais facilmente pela Internet. E por falar em Internet, alguém sabe me dizer como compactar um arquivo via página Web? Sim, e possível e não é nenhuma novidade eu sei, mas confesso que já recebi muitos e-mails de leitores e amigos questionando como fazer isso. Bem, em *PHP* é até bastante fácil, já que temos a disposição a biblioteca *ZLib*. Na versão de instalação do *PHP* para *Win32*, a

Resumo do DevMan

Como compactar arquivos utilizando Web Sites? Por incrível que pareça a Internet e conseqüentemente as linguagens têm evoluído bastante nos últimos tempos. Até bem pouco tempo era quase impossível compactar arquivos utilizando algum recurso de linguagem de programa para Web.

Hoje a realidade é outra. Com PHP podemos utilizar uma biblioteca nativa para fazer a compactação de qualquer tipo de arquivo em formato zip. Nesse artigo veremos como utilizar esse recurso na prática.

biblioteca *ZLib* vem previamente desabilitada, bastando alterar uma pequena linha no arquivo *PHP.INI*, veja:

```
;extension=php_zlib.dll
```

Basta retirar o ponto-e-vírgula do início da frase que a *DLL php_zlib.dll* está habilitada. Nesse artigo veremos como é simples fazer a compactação de arquivos on-line usando essa biblioteca. Mão na massa!

Considerações iniciais

Assim como já mencionei e importante habilitar a biblioteca *ZLib* no arquivo *php.ini*. Se você possui o *PHP* instalado em seu *Windows*, basta localizar esse arquivo no diretório *System32*, abrir no *Bloco de Notas* ou qualquer outro editor comum de textos e localizar a linha informada no início do artigo. Apenas retire o ponto-e-vírgula do início do arquivo de configurações e salve-o (**Figura 1**). Caso tenha instalado o programa Xampp em sua máquina, é possível saber se a biblioteca está habilitada acessando o link http://localhost/xampp e em seguida o link *phpinfo()*. Uma tabela com as funcionalidades do *ZLib* mostrará se está ativado (*enabled*) ou desativado (*disabled*), veja a **Figura 2**.

Desenvolvendo a aplicação

Uma informação importante e não muito agradável, é que não é tão fácil fazer uso da biblioteca *ZLib*, por isso vamos pegar carona e um *script* já pronto que está presente no *phpMyAdmin*: o *zip.lib.php*. Caso você tenha instalado o *phpMyAdmin*, basta localizar esse arquivo no diretório de instalação, copiá-lo para o diretório onde salvará sua aplicação e *PHP* e fazer uso dela através do método *require()*, como faremos mais adiante.

No meu caso é ainda mais fácil, pois tenho o *Xampp* e ele possui uma pré-instalação do *phpMyAdmin* em seu diretório. Veja na **Listagem 1** uma versão reduzida do código do *script zip.lib.php*.

Com podemos ver, na verdade *zip.lib.php* é uma classe que possui as funções necessárias para compactação de arquivos, o que me deixa bastante tranqüilo já que não preciso saber a fundo o que o *script* faz. Basicamente o *script* invoca as funções da *DLL* que por sua vez cria o arquivo *zip*.

Para montar nossa aplicação, criaremos um arquivo *HTML* que possibilitará escolhermos 3 arquivos por vez para fazer a compactação. Para isso abra seu editor preferido para que possamos digitar nosso código *HTML*. Eu usarei o *DreamWeaver* da *Adobe*.

Vamos iniciar desenvolvendo o layout da página principal, *index.html*. Digite o código da **Listagem 2** observando que estamos criando uma tela semelhante a **Figura 3.** Se você tiver familiaridade com o *Dreamweaver* ou qualquer outra ferramenta de desenvolvimento *Web*, ficará ainda mais fácil.

O código é simples, vejamos. Das *linhas 01* até *18* estamos apenas definindo o *layout* da página. Isso o próprio edito *HTML* já faz para nós, então vamos nos preocupar com as partes mais importantes. Se você conhece o mínimo de *HTML* verá que criamos um formulário usando a *tag* <*form*> na *linha 19*. Perceba que a ação (*action*) do nosso formulário está apontada para *compactar.php*. Muito bem, criaremos um arquivo *PHP* que receberá os arquivos selecionados e então os compactará. Em seguida, inserimos *3*

Nota do DevMan

Como compactar arquivos utilizando Web Sites? Por incrível que pareça a Internet e conseqüentemente as linguagens têm evoluído bastante nos últimos tempos. Até bem pouco tempo era quase impossível compactar arquivos utilizando algum recurso de linguagem de programa para Web.

O Xampp é um programa bastante utilizado pela comunidade PHP que cria um ambiente de testes para aplicações PHP. Quando instalado, o Xampp ativa um servidor Apache e se solicitado também um servidor MySQL. O PHP é instalado automaticamente junto com o software.

Para quem tem ou já teve dificuldades em criar um ambiente de testes do PHP no Windows, a dica é instalar esse programa que é free e muito fácil de usar. Para instalá-lo, basta efetuar o download da versão mais atual no site do desenvolvedor em www.apachefriends.org/en/xampp.html, seguir a instalação que não requer nenhum conhecimento adicional e logo ao final da instalação os scripts PHP já estarão disponíveis.

Mas atenção, o Xampp utiliza as portas 80 e 443 que são de Internet e SSL, respectivamente. O problema é que o IIS (Internet Information Services) também as usa, ou seja, é necessário manter o serviço do IIS parado enquanto efetuamos nossos testes utilizando o Xampp.

Na edição 101 de ClubeDelphi foi publicado um artigo QuickUpdate mostrando como fazer a instalação e configuração do Xampp em ambiente Windows, vale a pena dar uma olhada.

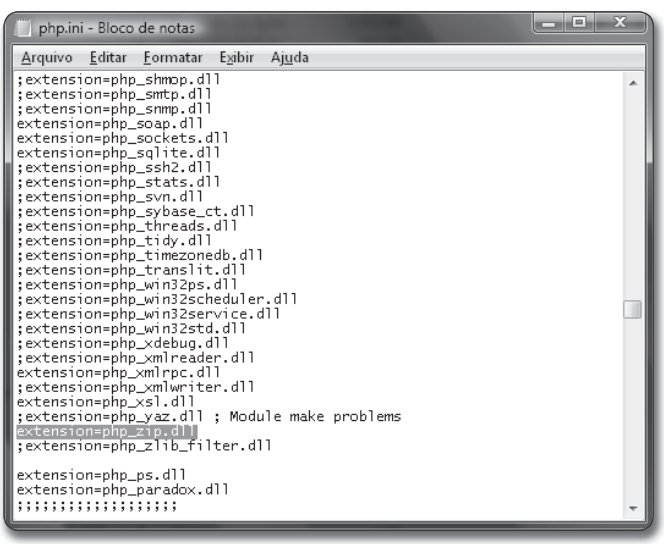


Figura 1. Configurando o arquivo php.ini

zlib

ZLib Support	enabled
Stream Wrapper support	compress.zlib://
Stream Filter support	zlib.inflate, zlib.deflate
Compiled Version	1.2.3
Linked Version	1.2.3

Directive	Local Value	Master Value
zlib.output_compression	Off	Off
zlib.output_compression_level	-1	-1
zlib.output_handler	*no value*	*no value*

Figura 2. Resultado do phpinfo() no Xampp

Listagem 1. Versão reduzida do código do script zip.lib.php

```
<?php
class zipfile
{
    var $datasec      = array();
    var $ctrl_dir     = array();
    var $eof_ctrl_dir = "\x50\x4b\x05\x06\x00\x00\x00\x00";
    var $old_offset   = 0;
    function unix2DosTime($unixtime = 0) {
        $timearray = ($unixtime == 0) ? getdate() :
         getdate($unixtime);

        if ($timearray['year'] < 1980) {
            $timearray['year']    = 1980;
            $timearray['mon']     = 1;
            $timearray['mday']    = 1;
            $timearray['hours']   = 0;
            $timearray['minutes'] = 0;
            $timearray['seconds'] = 0;
        }
        return (($timearray['year'] - 1980) << 25) |
          ($timearray['mon'] << 21) | ($timearray['mday'] << 16) |
          ($timearray['hours'] << 11) | ($timearray['minutes'] << 5) |
          ($timearray['seconds'] >> 1);
    }
    function addFile($data, $name, $time = 0)
    {
        $name     = str_replace('\\', '/', $name);

        $dtime    = dechex($this->unix2DosTime($time));
        $hexdtime = '\x' . $dtime[6] . $dtime[7]
                  . '\x' . $dtime[4] . $dtime[5]
                  . '\x' . $dtime[2] . $dtime[3]
                  . '\x' . $dtime[0] . $dtime[1];
        eval('$hexdtime = "' . $hexdtime . '";');

        $fr   = "\x50\x4b\x03\x04";
        $fr   .= "\x14\x00";
        $fr   .= "\x00\x00";
        $fr   .= "\x08\x00";
        $fr   .= $hexdtime;

        $unc_len = strlen($data);
        $crc     = crc32($data);
        $zdata   = gzcompress($data);
        $zdata   = substr(substr($zdata, 0, strlen($zdata) - 4),
                   2); // fix crc bug

        $c_len   = strlen($zdata);
        $fr      .= pack('V', $crc);
        $fr      .= pack('V', $c_len);
        $fr      .= pack('V', $unc_len);
        $fr      .= pack('v', strlen($name));
        $fr      .= pack('v', 0);
        $fr      .= $name;
        $fr .= $zdata;
        $this -> datasec[] = $fr;
        $cdrec = "\x50\x4b\x01\x02";
        $cdrec .= "\x00\x00";
        $cdrec .= "\x14\x00";
        $cdrec .= "\x00\x00";
        $cdrec .= "\x08\x00";
        $cdrec .= $hexdtime;
        $cdrec .= pack('V', $crc);
        $cdrec .= pack('V', $c_len);
        $cdrec .= pack('V', $unc_len);
        $cdrec .= pack('v', strlen($name));
        $cdrec .= pack('v', 0);
        $cdrec .= pack('v', 0);
        $cdrec .= pack('v', 0);
        $cdrec .= pack('v', 0);
        $cdrec .= pack('V', 32);
        $cdrec .= pack('V', $this -> old_offset);
        $this -> old_offset += strlen($fr);
        $cdrec .= $name;
        $this -> ctrl_dir[] = $cdrec;
    }
    function file()
    {
        $data    = implode('', $this -> datasec);
        $ctrldir = implode('', $this -> ctrl_dir);
        return
            $data .
            $ctrldir .
            $this -> eof_ctrl_dir .
            pack('v', sizeof($this -> ctrl_dir)) .
            pack('v', sizeof($this -> ctrl_dir)) .
            pack('V', strlen($ctrldir)) .
            pack('V', strlen($data)) .
            "\x00\x00";
    }

}
?>
```

Listagem 2. Código de layout da página principal

```
01 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
02 "http://www.w3.org/TR/html4/loose.dtd">
03 <html>
04 <head>
05 <meta http-equiv="Content-Type" content="text/html;
     charset=iso-8859-1">
06 <title>Conpactação de arquivos</title>
07 <style type="text/css">
08 <!--
09  .titulo {
10 font-family: Verdana, Arial, Helvetica, sans-serif;
11 font-size: 12px;
12 font-weight: bold;
13 }
14 -->
15 </style>
16 </head>
17 <body class="style1">
18 <p><span class="titulo">Script de Compacta&ccedil;&atilde;o de
     arquivos</span></p>
19 <form action="compactar.php" method="post" enctype="multipart/
     form-data"
20    name="form1">
21   <p> </p>
22   <p><span class="titulo">Arquivo 1:
23       </span>
24     <input name="file1" type="file" id="file1" size="50">
25     <br>
26     <span class="titulo">Arquivo 2:
27     </span>
28     <input name="file2" type="file" id="file2" size="50">
29     <br>
30     <span class="titulo">Arquivo 3:</span>
31     <input name="file3" type="file" id="file3" size="50">
32   </p>
33   <p>
34     <input name="btnCompactar" type="submit" id="btnCompactar"
         value="Compactar">
35 </p>
36   <p>     </p>
37 </form>
38 <p>
39 </body>
40 </html>
```

campos do tipo *file*. Para quem não sabe que tipo de campo é esse, são os campos que utilizamos para carregar arquivos ou fotos em sites como *Orkut* ou *GMail*. Pois bem, além desses campos inserimos também um botão que fará o *submit*, ou seja, enviará os dados coletados para a próxima página. Em execução, nossa página ficará bastante parecida com a **Figura 4**.

Feito isso, precisamos codificar a página *compactar.php*. Para isso crie um novo arquivo com esse nome e salve-o no diretório dos demais arquivos de *script*. Codifique-o conforme a **Listagem 3.** Agora vejamos o passo a passo da **Listagem 3.**

A primeira coisa que fazemos no *script compactar.php* é requerer o arquivo *zip.lib.php*. Quando usamos o método *require*, significa que requeremos o arquivo em questão, ou seja, ele precisa estar presente no diretório informado. Como não indicamos um diretório, ele deve estar junto da pasta onde o *script* atual está sendo executado, nesse caso o mesmo diretório de *compactar.php*. Em

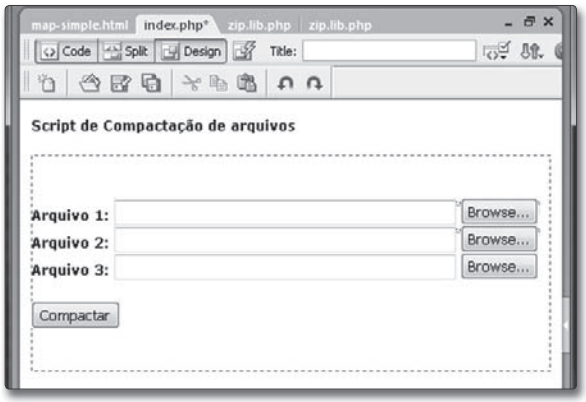


Figura 3. Formulário HTML para compactação

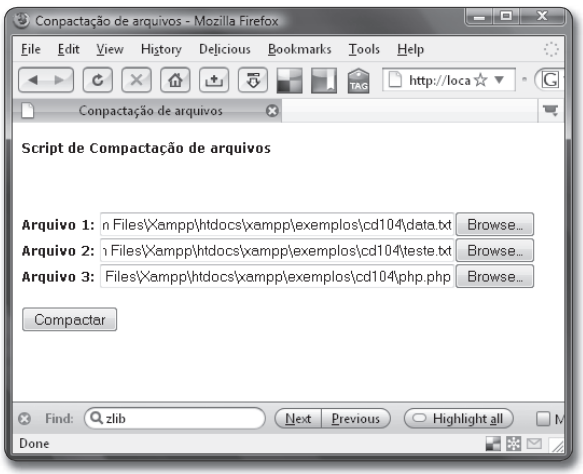


Figura 4. Página principal em execução

Figura 5. Confirmação de compactação dos arquivos

seguida, linhas 03, 04 e 05, recebemos o valor das variáveis *$file1*, *$file2* e *$file3*, que na verdade são o nomes dos três campos *file* do arquivo *index.html*. Quando usamos a variável global *$_GET* estamos pegando os dados que vieram pelo método *Get* enviados pela página anterior. Nas linhas seguintes, estamos abrindo cada arquivo recebido e armazenando nas variáveis *$com1*, *$com2* e *$com3*. Nas *linhas 15* até *19* estamos criando um objeto de compactação (*zipfile*) e adicionando os arquivos dentro dele. Quem efetivamente faz o trabalho de compactação é o método *fwrite*

Listagem 3. Código do script compactar.php

```
01 <?php
02   require("zip.lib.php");
03   $arq1 = $_GET["file1"];
04   $arq2 = $_GET["file2"];
05   $arq3 = $_GET["file3"];
06   $abre1 = fopen($arq1, "r");
07   $com1 = @fread($abre1, filesize($arq1));
08   @fclose($abre1);
09   $abre2 = fopen($arq2, "r");
10   $com2 = @fread($abre2, filesize($arq2));
11   @fclose($abre2);
12   $abre3 = fopen($arq3, "r");
13   $com3 = @fread($abre3, filesize($arq3));
14   @fclose($abre3);
15   $zip= new zipfile; //cria o objeto
16   $zip->addFile($com1,"test/$arq1"); //adiciona um arquivo ao zip
17   $zip->addFile($com2,"$arq2"); //adiciona um arquivo ao zip
18   $zip->addFile($com3,"test/int/$arq3"); //adiciona um
      arquivo ao zip
19   $strzip=$zip->file(); //string contendo o arquivo zip
20   $arq="back.zip";
21   print "<font face='Verdana' size='2' color='red'><b><center>
      <br><br><br>";
22   print "Os arquivos selecionados foram compactados com sucesso.";
23   print "</font>";
24   $abre = fopen($arq, "w");
25   $salva = @fwrite($abre, $strzip);
26   @fclose($abre);
27 ?>
```

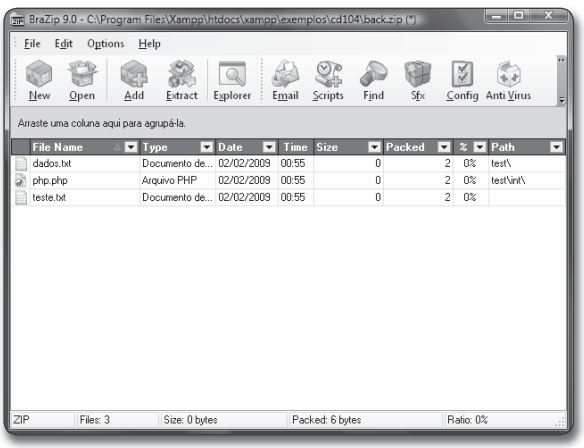


Figura 6. Resultado da compactação

presente na *linha 25* (**Figura 5**). Antes enviamos uma mensagem ao usuário dizendo que o processo correu bem (*Linha 22*).

Após o processo ter sido efetuado, os arquivos são *zipados* com o nome *back.zip* (*Linha 20*). Veja o resultado da compactação na **Figura 6**.

Conclusão

Compactar arquivos utilizando o *PHP* é muito simples como podemos ver nesse artigo. Além da biblioteca *ZLib* há outras bibliotecas capazes da mais variadas tarefas, tais como: emitir relatórios em PDF e criar gráficos. É importante estudar a fundo todas as funcionalidades do PHP, pois há uma infinidade de opções a serem utilizadas no dia-a-dia. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

A infra-estrutura de TI que você precisa.

A Alog trabalha 24 horas por dia, 7 dias por semana, 365 dias por ano para garantir aos seus clientes corporativos estabilidade, agilidade, redundância, escalabilidade, e mais do que tudo, **tranqüilidade**. Entre em contato e saiba por que a Alog é líder em hosting gerenciado

ALOG
DATA CENTERS DO BRASIL

HOSTING GERENCIADO | COLOCATION | CONTIGÊNCIA
E-MAIL CORPORATIVO | CONECTIVIDADE | SERVIÇOS PROFISSIONAIS

0800 282 3330 | alog.com.br

A INVASÃO DE PROPRIEDADE NÃO SE LIMITA A RESIDÊNCIAS.

Assegure-se que seu software está protegido pelo líder em segurança da informação, antes que seja tarde demais.

SafeNet tem à sua disposição a proteção de software mais avançada e segura. As Chaves de Hardware Sentinel oferecem tecnologia que incorpora criptografia de chave pública, criptografia AES e autenticação interna para garantir o mais alto nível de proteção anti-pirataria.

Peça seu Kit de Desenvolvimento hoje mesmo.

Entre em contato em +55 11 4208-7700 em SP-Brasil
ou por e-mail infopirataria@safenet-inc.com

25 YEARS
1983–2008

SafeNet